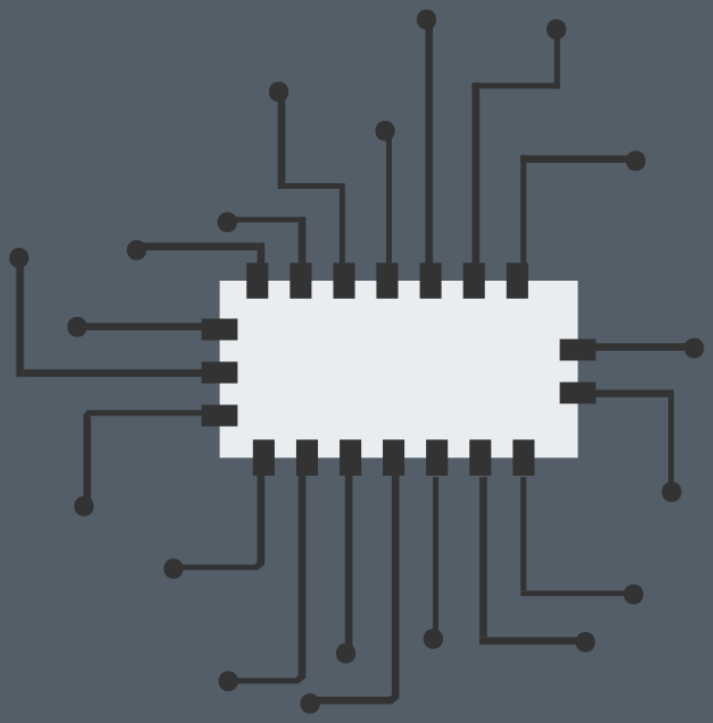
OBSTACLE AVOIDANCE CAR

AHMED HESHAM
ALAA HISHAM
HOSSAM ELWAHSH
SARAH MOHAMMED



Obstacle Avoidance Car Design

1. Project Introduction

This project aims to design a collision avoidance system for a four-wheel drive robot. By implementing intelligent sensing and control mechanisms, the system enables the robot to detect and avoid obstacles in its path, ensuring safe navigation.

1.1. Project Components

- ATmega32 microcontroller
- **Four** motors (**M1**, **M2**, **M3**, **M4**)
- One button to change default rotation direction (**PBUTTON0**)
- Keypad 3x3 (KPD1: start, KPD2: stop)
- One ultrasonic sensor
    - Vcc to **5v**
    - GND to GND
    - Trig to **PB3 (INT1)**
    - Echo to **PB2 (INT0)**
- LCD **2x16**

1.2. System Requirements

System Requirements:

1. The car **starts initially** from **0 speed**
2. The default rotation direction is to the **right**
3. Press (**Keypad Btn 1**), (**Keypad Btn 2**) to start or stop the robot respectively
4. **After Pressing Start:**
   - 4.1. The LCD will display a centered message in **line 1 "Set Def. Rot."**
   - 4.2. The LCD will display the selected option in **line 2 "Right"**
   - 4.3. The robot will wait for **5 seconds** to choose between **Right and Left**
     - 4.3.1. When **PBUTTON0** is pressed **once**, the default rotation will be **Left** and the **LCD line 2 will be updated**
     - 4.3.2. When **PBUTTON0** is pressed again, the default rotation will be **Right** and the **LCD line 2 will be updated**
     - 4.3.3. For each press the default rotation will changed and the **LCD line 2** is updated
     - **4.3.4. After the 5 seconds the default value of rotation is set (timeout)**
   - 4.4. The robot will move after **2 seconds** from setting the default direction of rotation.

5. For No obstacles or object is **far than 70 centimeters**:
   - 5.1. The robot will move **forward** with **30%** speed for **5 seconds**
   - 5.2. **After 5 seconds** it will move with **50%** speed **as long as** there was **no object** or objects are located at **more than 70 centimeters** distance
   - 5.3. The LCD will display the speed and moving direction **in line 1**:
     "Speed:00% Dir: F/B/R/S"
     F: forward, B: Backwards, R: Rotating, and S: Stopped
   - 5.4. The LCD will display Object distance in line 2 **"Dist.: 000 Cm"**

6. For Obstacles located **between 30 and 70 centimeters**
   - 6.1. The robot will **decrease** its **speed to 30%**
   - 6.2. LCD data is updated

7. For Obstacles located **between 20 and 30 centimeters**
   - 7.1. The robot will **stop** and **rotates 90 degrees** to **right/left** according to the chosen configuration
   - 7.2. The LCD data is updated

8. For Obstacles located **less than 20 centimeters**
   - 8.1. The robot will **stop**, **move backwards** with **30% speed until distance is greater than 20 and less than 30**
   - 8.2. The LCD data is updated
   - 8.3. Then perform **point 8**

9. Obstacles surrounding the robot (Bonus)
    9.1. If the robot rotated for **360 degrees** without finding any distance greater **than 30** it will **stop**
    9.2. LCD data will be updated.
    9.3. The robot will frequently **(each 3 seconds)** check if any of the obstacles was removed or not and move in the direction of the furthest object

1.3. Assumptions

- 4WD Robot Specifications - 4WD Complete Mini Plastic Robot Chassis Kit
- For the Robot to rotate in place around its pivot point we calculated that:
    - Left motors going forward, Right motors going backward
    - Rotation frequency: **488 Hz**
    - Rotation duration: **650 ms**
    - Rotation duty cycle: **50%**

2. High Level Design

2.1. System Architecture

2.1.1. Definition

Layered Architecture (*Figure 1*) describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software.

Microcontroller Abstraction Layer (*MCAL*) is a software module that directly accesses on-chip MCU peripheral modules and external devices that are mapped to memory, and makes the upper software layer independent of the MCU.

Hardware Abstraction Layer (*HAL*) is a layer of programming that allows a computer OS to interact with a hardware device at a general or abstract level rather than at a detailed hardware level.

2.1.2. Layered Architecture

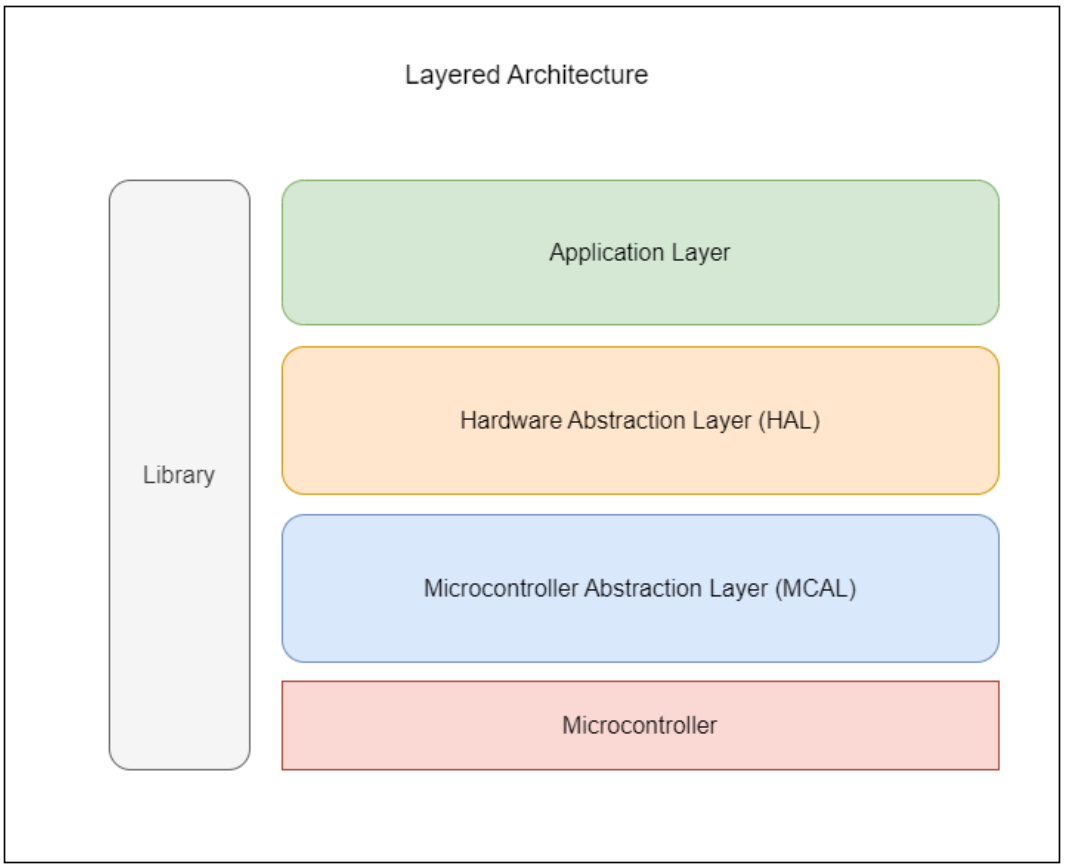
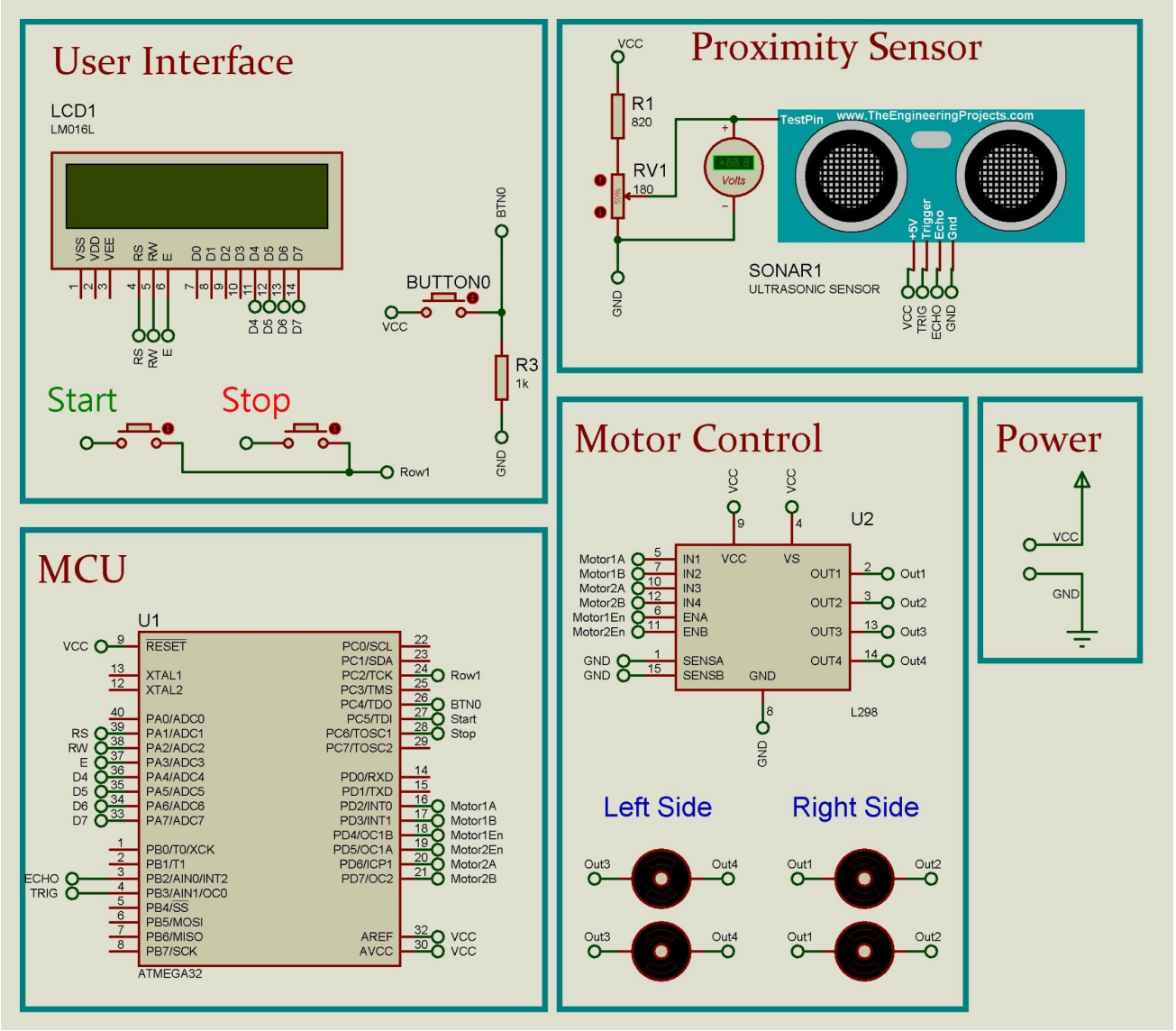


Figure 1. Layered Architecture Design

2.1.3. Project Schematic



2.2. Modules Description

2.2.1. DIO (Digital Input/Output) Module

The *DIO* module is responsible for reading input signals from the system's sensors (such as buttons) and driving output signals to the system's actuators (such as *LEDs*). It provides a set of APIs to configure the direction and mode of each pin (input/output, pull-up/down resistor), read the state of an input pin, and set the state of an output pin.

2.2.2. EXI Module

The *EXI* (External Interrupt) module is responsible for detecting external events that require immediate attention from the microcontroller, such as a button press. It provides a set of APIs to enable/disable external interrupts for specific pins, set the interrupt trigger edge (rising/falling/both), and define an interrupt service routine (*ISR*) that will be executed when the interrupt is triggered.

2.2.3. TIMER Module

The *TIMER* module is responsible for generating timing events that are used by other modules in the system. It provides a set of APIs to configure the timer clock source and prescaler, set the timer mode (count up/down), set the timer period, enable/disable timer interrupts, and define an ISR that will be executed when the timer event occurs.

2.2.4. ICU Module

The ICU Module is a software component designed to interface with an Ultrasonic sensor in our project. It enables the accurate detection of the distance between the car and surrounding objects in all directions. By leveraging input capture techniques, the ICU Module captures the sensor's readings and provides real-time distance measurements. This crucial information aids in implementing a comprehensive collision avoidance system, ensuring safe navigation for the vehicle in various scenarios.

2.2.5. LCD Module

LCD stands for "Liquid Crystal Display," which is a type of flat-panel display used in electronic devices to display text and graphics. We're using the 4-bit mode to reduce the number of I/O pins needed to interface with the LCD. The module includes a controller, a display, and a backlight. By interfacing with the LCD module and writing the necessary code, we're able to provide the user with real-time information about the current speed and direction of the car as well as providing an interface to set the default rotation direction.

2.2.6. BTN Module

The *BTN* (Button) module is responsible for reading the state of the system's buttons. It provides a set of APIs to enable/disable button interrupts, set the button trigger edge (rising/falling/both), and define an ISR that will be executed when a button press is detected.

2.2.7. DCM Module

The *DCM* (DC Motor) module is responsible for controlling the speed and direction of the system's DC motors. It provides a set of APIs to set the speed and direction of each motor, and to stop all motors. It also uses the *TIMER* module to generate *PWM* (Pulse Width Modulation) signals that control the motor speed.

2.2.8. KPD Module

Keypad is an analog switching device which is generally available in matrix structure. It is used in many embedded system applications for allowing the user to perform a necessary task. A matrix Keypad consists of an arrangement of switches connected in matrix format in rows and columns. The rows and columns are connected with a microcontroller such that the rows of switches are connected to one pin and the columns of switches are connected to another pin of a microcontroller.

2.2.9. Delay Module

Delay is a software-implemented module that uses a normal timer to generate both non-blocking delays and blocking delays. it also sends to the timer call back function to be called when the timer interrupt occurs.

2.2.10. PWM Module

PWM is a software-implemented module that uses a normal timer to generate pulse width module signals through chosen pins. It has the accessibility to edit the duty cycle of the output signal too.

2.2.11. Ultrasonic Module

The Ultrasonic module is a key component in our project for distance sensing. It utilizes ultrasonic waves to measure distances between the module and surrounding objects. By emitting ultrasonic pulses and calculating the time taken for the echoes to return, the module provides accurate distance measurements. Its compact size, low power consumption, and wide detection range make it an ideal choice for collision avoidance applications. With its reliable performance and easy integration, the Ultrasonic module enhances the safety and precision of our project's distance sensing capabilities.

2.2.12. Design

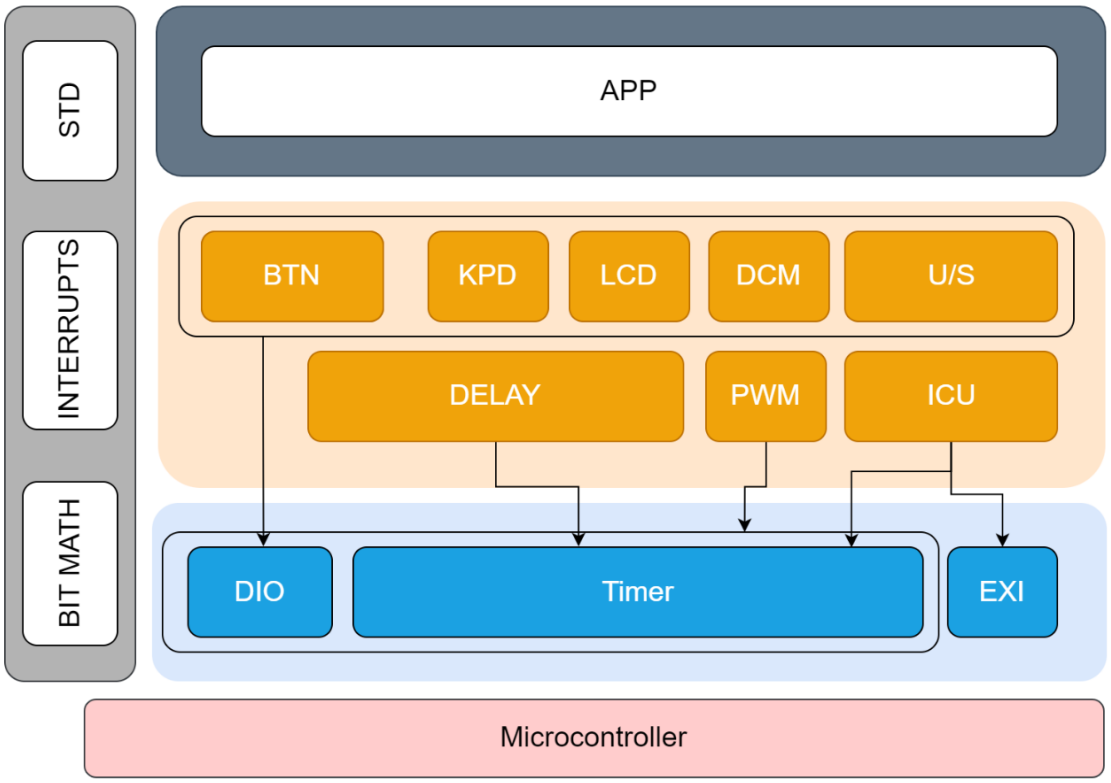


Figure 3. System Modules Design

2.3. Drivers' Documentation (APIs)

2.3.1 Definition

An *API* is an *Application Programming Interface* that defines a set of *routines*, *protocols* and *tools* for creating an application. An *API* defines the high level interface of the behavior and capabilities of the component and its inputs and outputs.

An *API* should be created so that it is generic and implementation independent. This allows for the API to be used in multiple applications with changes only to the implementation of the API and not the general interface or behavior.

2.3.2. MCAL APIs

2.3.2.1. DIO Driver

```
|
|  Function to set the direction of a given port
|
| This function takes an 8-bit value and sets the direction of each
| pin in the given port according to the corresponding bit value
|
| Parameters
|          [in] en_a_port    The port to set the direction of
|          [in] u8_a_portDir The desired port direction
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPortDir(en_DIO_port_t en_a_port,  u8 u8_a_portDir);
|
|  Function to set the value of a given port
|
| This function takes an 8-bit value and sets the value of each
| pin in the given port according to the corresponding bit value
|
| Parameters
|          [in] en_a_port    The port to set the value of
|          [in] u8_a_portVal The desired port value
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPortVal(en_DIO_port_t en_a_port,  u8 u8_a_portVal);
```

```
|
|  Function to set the direction of a given pin
|
| This function takes an en_DIO_pinDir_t value and sets the direction
| of the given pin accordingly
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to set direction of
|          [in] en_a_pinDir The desired pin direction (INPUT/OUTPUT)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinDir (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_pinDir_t en_a_pinDir);

|
|  Function to set the value of a given pin
|
| This function takes an en_DIO_level_t value and sets the value
| of the given pin accordingly
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to set value of
|          [in] en_a_pinDir The desired pin value (HIGH/LOW)
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_setPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
en_DIO_level_t en_a_pinVal);
```

```
|
|  Function to toggle the value of a given pin
|
| If the pin value is high, this function sets it to low
| and if it is low it sets it to high
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to toggle value of
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_togPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin);

|
|  Function to get the value of a given pin
|
| This function reads the value of the given pin and
| returns the value in the given address
|
| Parameters
|          [in] en_a_port    The port of the desired pin
|          [in] en_a_pin     The desired pin to read value of
|          [out] pu8_a_Val   address to return the pin value into
|
|
| Return
|       en_DIO_error_t value that indicates operation success/failure
|               (DIO_OK in case of success or DIO_ERROR in case of failure)
|
en_DIO_error_t DIO_getPinVal (en_DIO_port_t en_a_port, en_DIO_pin_t en_a_pin,
u8* pu8_a_Val);
```

2.3.2.2. EXI Driver

```
|
| Initializes given EXI as configured
|
| This function initializes the passed interrupt with the configured
| parameters in the configuration source file
|
| Parameters
|          [in] en_a_IntNumber the interrupt to be initialized
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_init(en_EXI_num_t en_a_intNumber);

|
|  Function to choose the trigger event for given EXI
|
| This function sets the given EXI to be triggered whenever
| an event that matches the given sense mode occurs
|
| Parameters
|          [in] en_a_IntNumber The interrupt to be configured
|          [in] en_a_SenseMode The event to trigger the EXI
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_setSense(en_EXI_num_t en_a_intNumber, en_EXI_senseMode_t
en_a_senseMode);
```

```
|
| Function to enable/disable given EXI
|
| This function sets or clears the specific interrupt enable bit
| for the given interrupt to enable or disable it
|
| Parameters
|          [in] en_a_IntNumber The interrupt to be configured
|          [in] en_a_intState  EXI state (EXI_ENABLE/EXI_DISABLE)
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_setState(en_EXI_num_t en_a_IntNumber, en_EXI_state_t
en_a_intState);

|
| Function to set a function to call when EXI is triggered
|
| This function sets a callback function to be called whenever
| the given interrupt is triggered
|
| Parameters
|          [in] en_a_IntNumber The desired EXI number
|          [in] pv_a_Function  The function to call
|
|
| Return
|       en_EXI_error_t value that indicates operation success/failure
|               (EXI_OK in case of success or EXI_ERROR in case of failure)
|
en_EXI_error_t EXI_setCallback(en_EXI_num_t en_a_IntNumber, void
(*pv_a_Function)(void));
```

2.3.2.3. TIMER Driver

```
|
| Syntax          : en_TIMER_error_t TIMER_init( void )
| Description     : Initialize Timer according to preprocessed
|                    configured definitions
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value    : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
```

```
en_TIMER_error_t TIMER_init( void );
```

```
|
|  Syntax          : en_TIMER_error_t TIMER_setTime
|                    (en_TIMER_number_t en_a_timerUsed, f32 f32_a_desiredTime)
|  Description     : set the time at which the timer interrupts
|  Sync\Async      : Synchronous
|  Reentrancy      : Reentrant
|  Parameters (in) : en_TIMER_number_t            en_a_timerUsed
|                    f32                          f32_a_desiredTime
|  Parameters (out): None
|  Return value:   : en_TIMER_error_t             TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
```

```
en_TIMER_error_t TIMER_setTime(en_TIMER_number_t en_a_timerUsed, f32 f32_a_desiredTime);
```

```
|
|  Syntax          : en_TIMER_error_t TIMER_pwmGenerator
|             (en_TIMER_number_t en_a_timerUsed , u16 u16_a_onTime, u16 u16_a_offTime)
|  Description     : initialize the timer to generates pwm signal using
|                    normal mode
|  Sync\Async      : Synchronous
|  Reentrancy      : Reentrant
|  Parameters (in) : en_TIMER_number_t            en_a_timerUsed
|                      u16                        u16_a_onTime
|                      u16                        u16_a_offTime
|  Parameters (out): None
|  Return value:   : en_TIMER_error_t             TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
```

```
en_TIMER_error_t TIMER_pwmGenerator(en_TIMER_number_t en_a_timerUsed , u16 u16_a_onTime, u16
u16_a_offTime);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_resume(en_TIMER_number_t en_a_timerUsed)
| Description     : makes the timer to start/resume counting
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
en_TIMER_error_t TIMER_resume(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_reset(en_TIMER_number_t en_a_timerUsed)
| Description     : makes the timer to reset counting from the beginning
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
en_TIMER_error_t TIMER_reset(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_getElapsedTime
|                     (en_TIMER_number_t en_a_timerUsed, u32* u32_a_elapsedTime)
| Description     : returns the elapsed time since the timer started
|                    from the beginning in microseconds
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): u32                           u32_a_elapsedTime
| Return value:   : en_TIMER_error_t              TIMER_OK = 0
|                                                 TIMER_WRONG_TIMER_USED = 1
|                                                 TIMER_WRONG_DESIRED_TIME = 2
|                                                 TIMER_NOK = 3
|
en_TIMER_error_t TIMER_getElapsedTime(en_TIMER_number_t en_a_timerUsed, u32* u32_a_elapsedTime);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_pause(en_TIMER_number_t en_a_timerUsed)
| Description     : makes the timer to pause counting
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                           TIMER_WRONG_TIMER_USED = 1
|                                           TIMER_WRONG_DESIRED_TIME = 2
|                                           TIMER_NOK = 3
|
en_TIMER_error_t TIMER_pause(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_disableInterrupt(en_TIMER_number_t en_a_timerUsed)
| Description     : Disables timer's interrupts
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                           TIMER_WRONG_TIMER_USED = 1
|                                           TIMER_WRONG_DESIRED_TIME = 2
|                                           TIMER_NOK = 3
|
en_TIMER_error_t TIMER_disableInterrupt(en_TIMER_number_t en_a_timerUsed);

|
| Syntax          : en_TIMER_error_t TIMER_enableInterrupt(en_TIMER_number_t en_a_timerUsed)
| Description     : Enables timer's interrupts
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
| Parameters (out): None
| Return value:   : en_TIMER_error_t        TIMER_OK = 0
|                                           TIMER_WRONG_TIMER_USED = 1
|                                           TIMER_WRONG_DESIRED_TIME = 2
|                                           TIMER_NOK = 3
|
en_TIMER_error_t TIMER_enableInterrupt(en_TIMER_number_t en_a_timerUsed);
```

```
|
| Syntax          : en_TIMER_error_t TIMER_setCallBack
|                     (en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void))
| Description     : sets the call back function for a specific timer
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                    void                         (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setCallBack(en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void));

|
| Syntax          : en_TIMER_error_t TIMER_setDelayTime
|                                    (en_TIMER_number_t en_a_timerUsed, f32 f32_a_timeInMS)
| Description     : Set delay time for blocking delay using a specific timer
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                      f32                        f32_a_timeInMS
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setDelayTime(en_TIMER_number_t en_a_timerUsed, f32 f32_a_timeInMS);

|
| Syntax          : en_TIMER_error_t TIMER_setPwmOnCallBack
|                                    (en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void))
| Description     : Set callback function for the task done while signal is high
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t             en_a_timerUsed
|                    void                         (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setPwmOnCallBack(en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void));
```

```
|
| Syntax          : en_TIMER_error_t TIMER_setPwmOffCallBack
|                                    (en_TIMER_number_t en_a_timerUsed, void (*funPtr)(void))
| Description     : Set callback function for the task done while signal is low
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_TIMER_number_t              en_a_timerUsed
|                     void                         (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_TIMER_error_t       TIMER_OK = 0
|                                          TIMER_WRONG_TIMER_USED = 1
|                                          TIMER_WRONG_DESIRED_TIME = 2
|                                          TIMER_NOK = 3
|
en_TIMER_error_t TIMER_setPwmOffCallBack(en_TIMER_number_t en_a_timerUsed, void
(*funPtr)(void));
```

2.3.3. HAL APIs

2.3.3.1. LCD APIs

```
 | function         : LCD_vidInit
 | description      : func to set LCD initialization
 | input param      : void
 | return           : void

void LCD_vidInit(void);


 | function         : LCD_vidWritecmd
 | description      : func to configure some commands on lcd
 | input param      :
 |u8commandCopy --> take lcd cmd instructions from instruction table
<https://components101.com/sites/default/files/component_datasheet/16x2%20LCD%20Datash
eet.pdf>
 | return           : void

void LCD_vidWritecmd(u8 u8commandCopy);


 | function         : HLCD_vidWriteChar
 | description      : func to write char on lcd
 | input param      : u8CharCopy -> take ascii code of char   or   char address on
CGROM
 | return           : void

void LCD_vidWriteChar(u8 u8CharCopy);

 | function         : LCD_ClrDisplay
 | description      : func to clear anything on lcd
 | input param      : void
 | return           : void

void LCD_ClrDisplay(void);



 | function         : HLCD_ShiftLeft
 | description      : func to shift the lcd display from right to left
 | input param      : void
 | return           : void
 void LCD_ShiftLeft(void);
```

```
 | function         : LCD_gotoXY
 | description      : func to determine position which char print at this position on
lcd  ### NOTE : (2rows x 16coloms)
 | input param      :  row -> take row number 0 or 1
 |                     pos -> take colom number from 0 ~ 15
 | return           : void

void LCD_gotoXY(u8 row, u8 pos);



 | function         : LCD_WriteString
 | description      : func to write string on lcd
 | input param      : str --> which take string as argument
 | return           : void

void LCD_WriteString(u8* str);


 | function         : LCD_WriteInt
 | description      : func to write integer number on lcd
 | input param      : number --> which take number as argument
 | return           : void
 void LCD_WriteInt(u32 number);


 | function         : LCD_WriteArabic
 | description      : func to write Arabic string on lcd
 | input param      : u8ArCharCopy --> which take string as argument
 | return           : void

void LCD_WriteArabic(u8 u8ArCharCopy);



 | function         : LCD_vidCreatCustomChar
 | description      : func to store new patterm on CGRAM
 | input param      :
 |                         pu8custom  -> take pointer to array which having LCD
Custom Character Generated data ### take only 8 characters
 |                         u8Location -> determine location on CGRAM [0 ~ 8]
 | return           : void

void LCD_vidCreatCustomChar(u8* pu8custom, u8 u8Location);
```

2.3.3.2. BTN APIs

```
|  description     : func to initialize button
| Parameters
|             u8_a_buttonPort:  read port number.
|             u8_a_buttonPin :  read pin number .
| Return
|     en_buttonError_t if the button state was read successfully,

en_buttonError_t BUTTON_init(u8 u8_a_buttonPort, u8 u8_a_buttonPin);

|  description     : func to read button
| Parameters
|             u8_a_buttonPort:  read port number.
|             u8_a_buttonPin :  read pin number .
|             u8_a_buttonState: pointer to read button state
| Return
|     en_buttonError_t if the button state was read successfully,

en_buttonError_t BUTTON_read(u8 u8_a_buttonPort, u8 u8_a_buttonPin, u8
*u8_a_buttonState);
```

2.3.3.3. DCM APIs

```
|
| Syntax          : en_DCM_error_t DCM_init (void)
| Description     : Initializes DCM module
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DCM_error_t                DCM_OK = 0
|                                                  DCM_NOK = 1
|
en_DCM_error_t DCM_init                  (void);
```

```
| Syntax          : en_DCM_error_t DCM_setDirection
|       (en_DCM_number_t en_a_dcmNumber, en_DCM_direction_t en_a_direction)
| Description     : Sets Directions for the a specific DCM
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : en_DCM_number_t          en_a_dcmNumber
|                   en_DCM_direction_t       en_a_direction
| Parameters (out): None
| Return value:   : en_DCM_error_t                 DCM_OK = 0
|                                                  DCM_NOK = 1
|
en_DCM_error_t DCM_setDirection     (en_DCM_number_t en_a_dcmNumber,
en_DCM_direction_t en_a_direction);

|
| Syntax          : en_DCM_error_t DCM_speed (u8 u8_a_speed)
| Description     : Sets speed for DCMs
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                                u8_a_speed
| Parameters (out): None
| Return value:   : en_DCM_error_t                 DCM_OK = 0
|                                                  DCM_NOK = 1
|
en_DCM_error_t DCM_speed                  (u8 u8_a_speed);

|
| Syntax          : en_DCM_error_t DCM_start (void)
| Description     : Starts DCMs to rotate
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DCM_error_t                 DCM_OK = 0
|                                                  DCM_NOK = 1
|
en_DCM_error_t DCM_start                  (void);

|
| Syntax          : en_DCM_error_t DCM_stop (void)
| Description     : Stops DCMs from rotating
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DCM_error_t                 DCM_OK = 0
|                                                  DCM_NOK = 1
|
en_DCM_error_t DCM_stop                   (void);
```

2.3.3.4. KPD APIs

```
 |Name : KEYPAD_init()
 |Description : This Function Initializes keypad pins (Rows are outputs & Columns are
inputs).
 | ARGS : void
 |return : void
void KEYPAD_init(void);



 |Name : KEYPAD_GetButton
 |Description : This function gets keypad value pressed or not pressed.
 |ARGS : void
 |return : u8 -> value of keypad
u8 KEYPAD_GetButton(void);
```

2.3.3.5. ICU APIs

```
| Initializes the ICU driver
|
| This function initializes a software ICU driver from ICU configuration,
| init echo pin as input, uses timer to calculate elapsed time by a signal
| being high until it goes back low
|
| Parameters
|            [in]u8_a_echoPin I/O pin number to receive echoed signal
|            [in]cf_a_timeReceived callback function to send elapsed duration
| Return
|     None
|
void ICU_init(void);

| Resets and starts the ICU algorithm to capture the elapsed time duration starting
| now until the monitored pin signal is brought low again
|
| Return
|     None
|
void ICU_getCaptureValue(void);
```

2.3.3.6. Ultrasonic APIs

```
| Initializes the ultrasonic driver
|
| Parameters
|            [in]u8_a_triggerPin I/O pin number to send trigger signal
|            [in]u8_a_echoPin I/O pin number to receive echoed signal
|
| Return
|     None
|
void US_init(u8 u8_a_triggerPin, u8 u8_a_echoPin);

| Initiates a get distance request
|
| This function sends a signal out to the trigger pin, waits for echo signal
| to come back and finally calculates the distance the signal traveled using
| the elapsed time duration used by the signal to arrive back on the echo pin
|
| Return
|     float distance (in mm)
|
float US_getDistance(void);

| Event Handler: called when echo time is received from ICU (input capture unit)
|
| This function is called by the ICU as an event callback when the trigger signal
| is received back on the echo pin, the function receives the elapsed time taken.
|
| Parameters
|       [in]u8_a_timeElapsed time elapsed (duration) by trigger signal to echo back
|
| Return
|     None
|
void US_evtEchoTimeReceived(u8 u8_a_timeElapsed);
```

2.3.3.7. Delay APIs

```
|
| Syntax          : en_DELAY_error_t DELAY_init (void)
| Description     : Initializes delay module
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_init   (void);

|
| Syntax          : en_DELAY_error_t DELAY_setTime (f32 f32_a_timeInMS)
| Description     : Sets blocking delay time
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : f32                       f32_a_timeInMS
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_setTime (f32 f32_a_timeInMS);

|
| Syntax          : en_DELAY_error_t DELAY_setTime (f32 f32_a_timeInMS)
| Description     : Sets non-blocking delay time
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : f32                       f32_a_timeInMS
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_setTimeNonBlocking (f32 f32_a_timeInMs);

|
| Syntax          : en_DELAY_error_t DELAY_setCallBack (void (*funPtr)(void))
| Description     : Sets call back for the function which gonna be performed
|                   when the nonblocking delay times out
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : void                      (*funPtr)(void)
| Parameters (out): None
| Return value:   : en_DELAY_error_t          DELAY_OK = 0
|                                             DELAY_NOK = 1
|
en_DELAY_error_t DELAY_setCallBack (void (*funPtr)(void));
```

2.3.3.8. PWM APIs

```
|
| Syntax          : en_PWM_error_t PWM_init (void)
| Description     : Initializes PWM module
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_init (void);

|
| Syntax          : en_PWM_error_t PWM_setDutyCycle (u8 u8_a_speed)
| Description     : Sets PWM duty cycle
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : u8                              u8_a_speed
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_setDutyCycle (u8 u8_a_speed);

|
| Syntax          : en_PWM_error_t PWM_start (void)
| Description     : Starts PWM
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_start (void);

|
| Syntax          : en_PWM_error_t PWM_stop (void)
| Description     : Stops PWM
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
en_PWM_error_t PWM_stop (void);
```

```
|
| Syntax          : en_PWM_error_t PWM_onTask (void)
| Description     : task called by the interrupt when the signal is in HIGH phase
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
void PWM_onTask (void);

|
| Syntax          : en_PWM_error_t PWM_onTask (void)
| Description     : task called by the interrupt when the signal is in LOW phase
| Sync\Async      : Synchronous
| Reentrancy      : Reentrant
| Parameters (in) : None
| Parameters (out): None
| Return value:   : en_PWM_error_t                  PWM_OK = 0
|                                                   PWM_NOK = 1
|
void PWM_offTask (void);
```

2.3.4. APP APIs

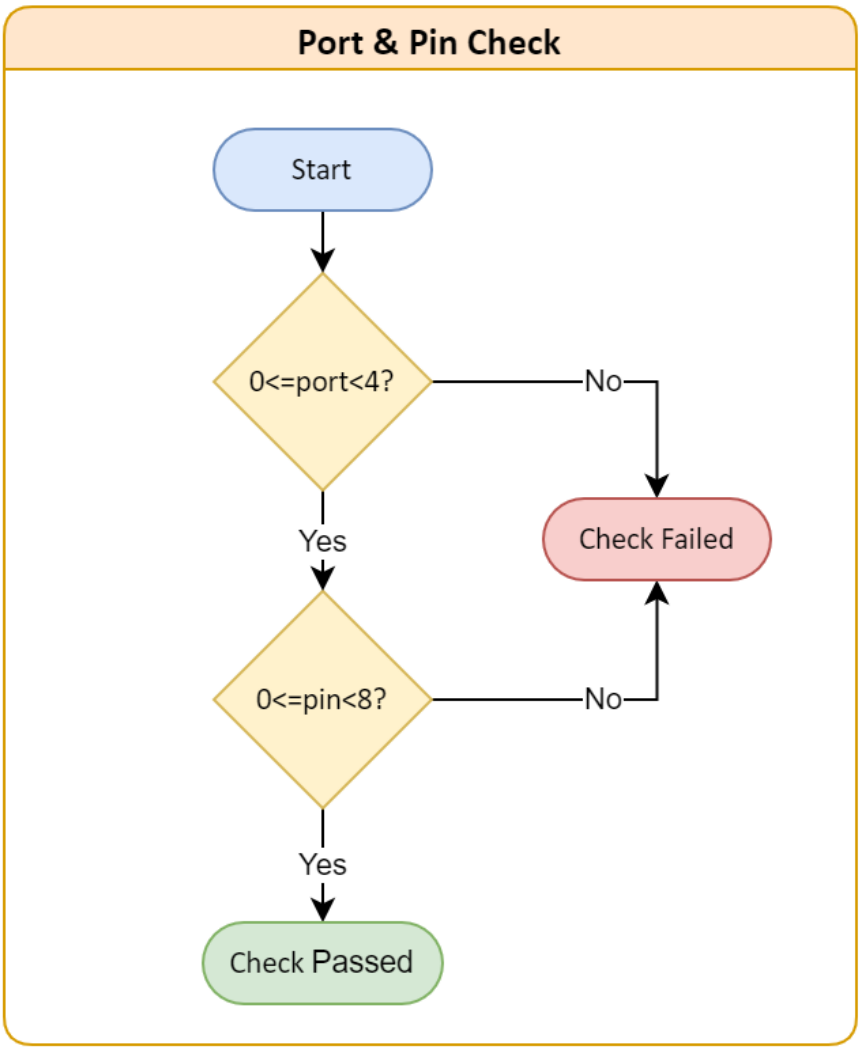
```
| Updates LCD (UI) with current speed (global), current direction (Fwd,
|  Bck, Rot, Stp)
| and current distance from obstacle in front of the robot
|
| Parameters
|           [in]u16_a_dist distance between the robot and the obstacle
|                          in-front of it in CM
| Return
|       void
|
static void APP_updateUI(u16 u16_a_dist);

| Initializes the application by initializing MCAL and HAL.
| This function initializes the General Interrupt Enable (GIE), sets up
|  callback functions
|
| Return
|      void
|
void APP_initialization(void);

| This function starts the car program and keeps it running indefinitely.
| The function uses a while loop to continuously check for the required
|  app mode.
| The app mode is checked using a switch statement, which contains various
|  cases that correspond to the different modes of operation for the car program.
| Each case contains a series of steps to be executed to perform the desired
|  action for that mode.
|
| Return
|      void
|
void APP_startProgram(void);

| Used to switch between app states to initialize standard UI elements
| before main app flow (loop)
|
| Parameters
|            [in] u8_a_state state to set (APP_STATE_LAUNCH, APP_STATE_...)
|
| Return
|      void
|
static void APP_switchState(u8 u8_a_state);

| callback for delay
void APP_delayNotification(void);
```

3. Low Level Design

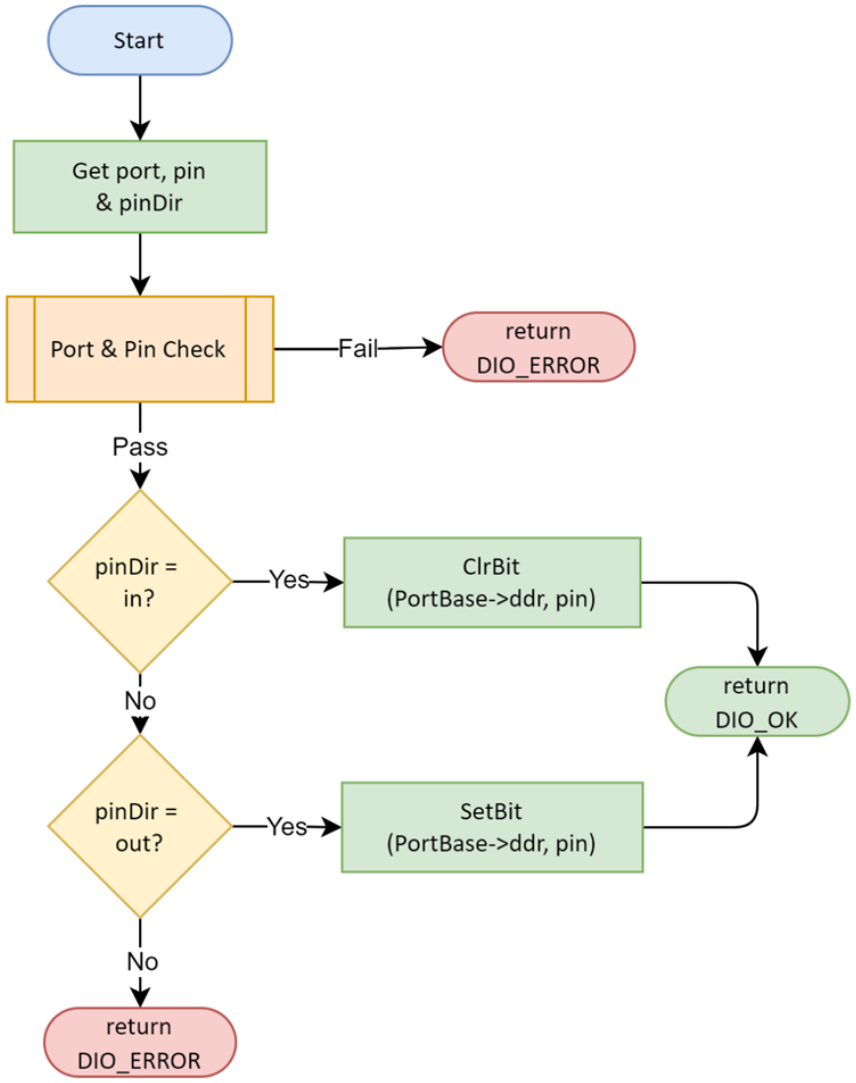
3.1. MCAL Layer

3.1.1. DIO Module

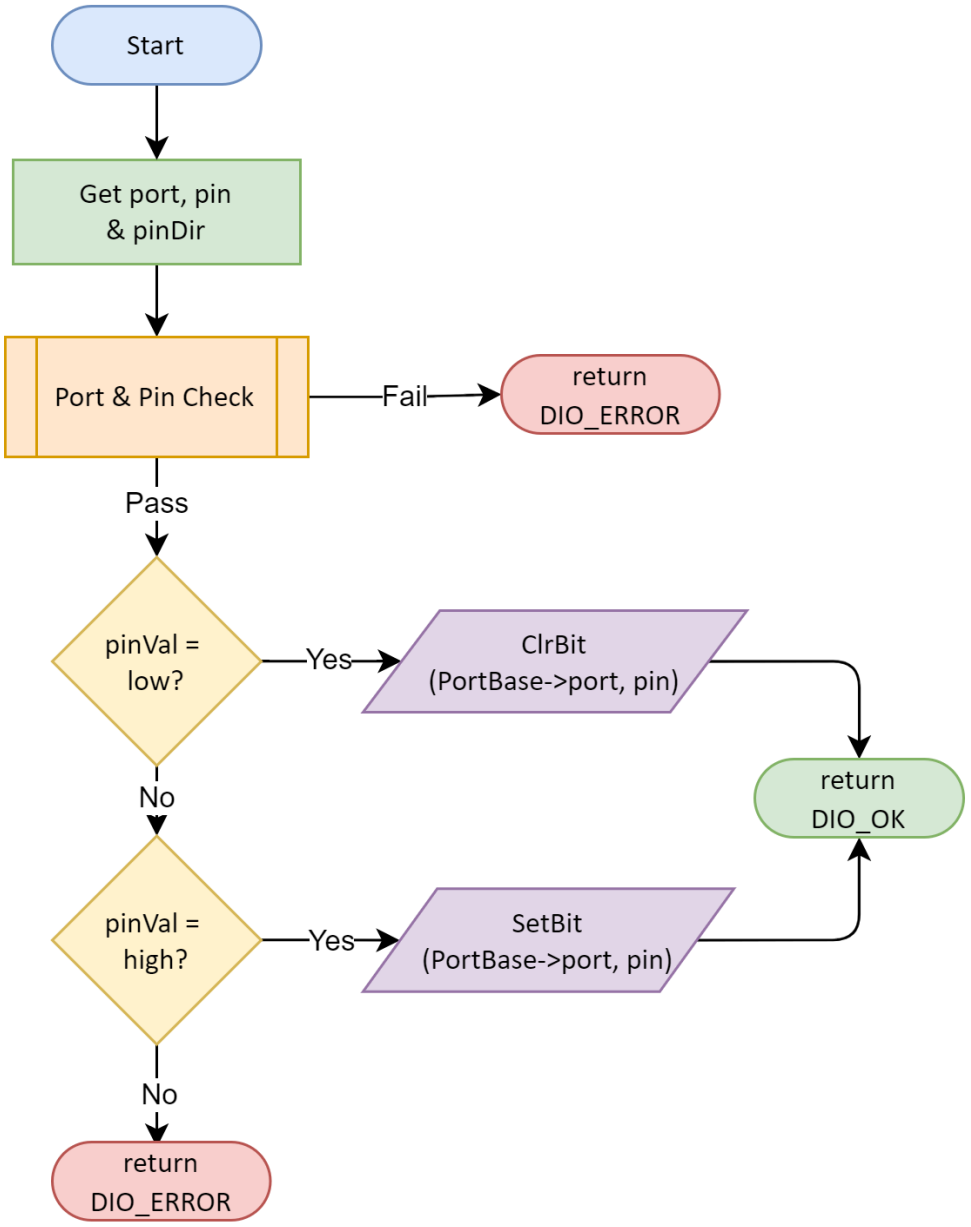
3.1.1.a. sub process



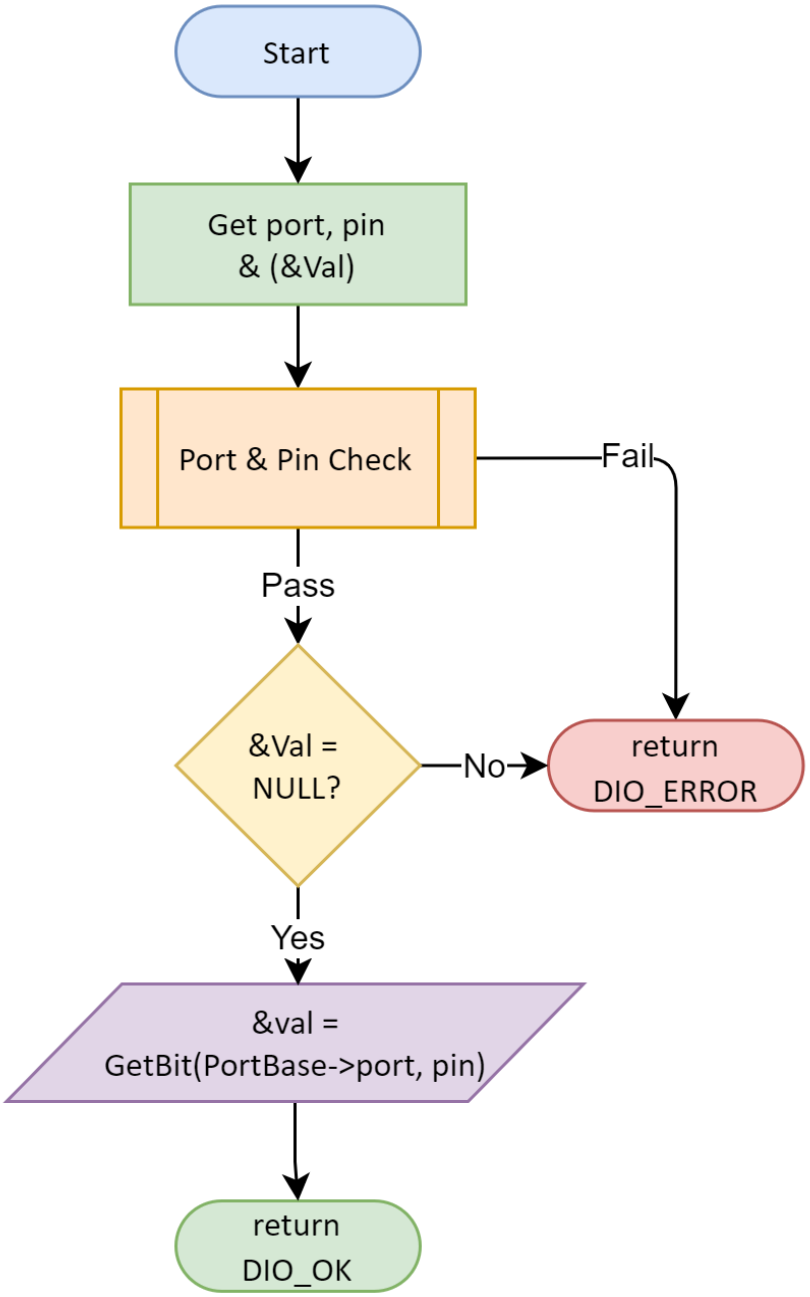
3.1.1.1. DIO_setPinDir



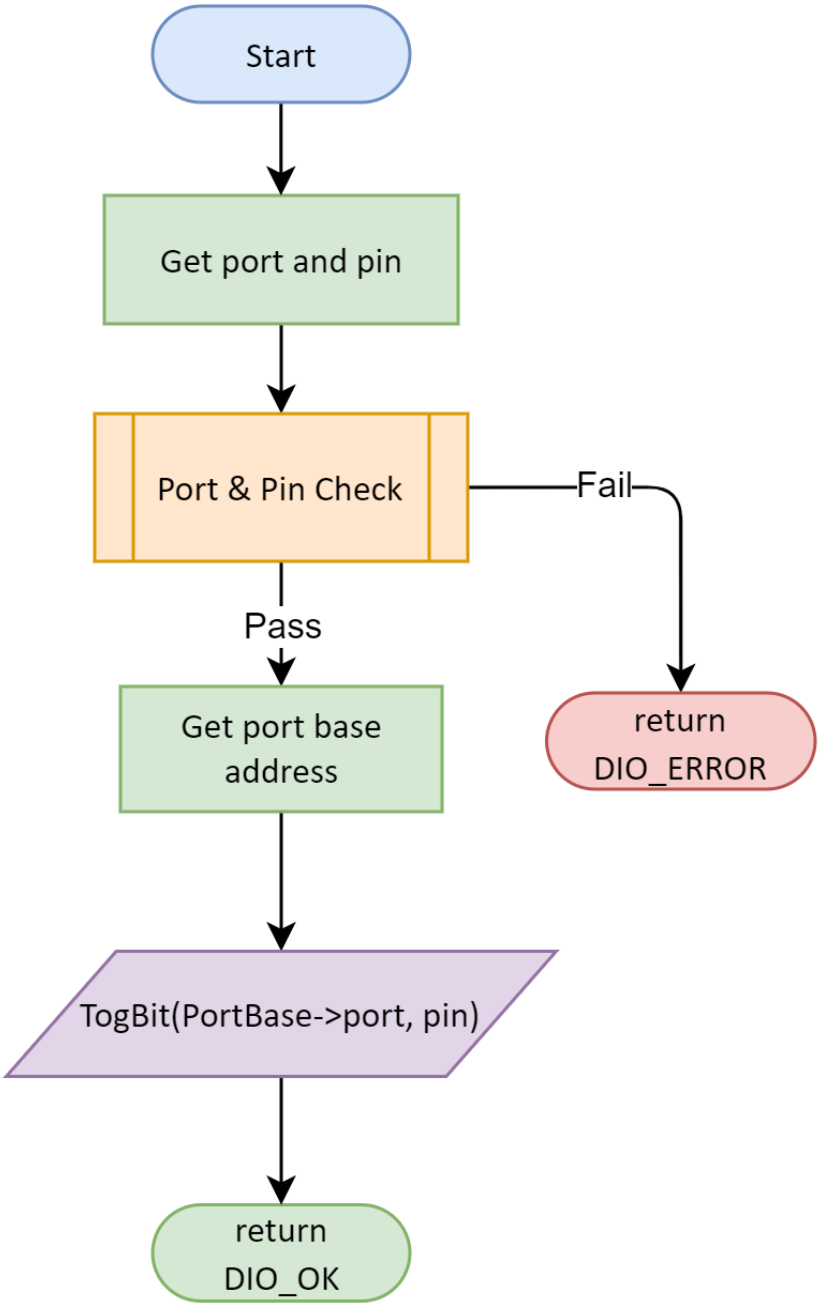
3.1.1.2. DIO_setPinVal



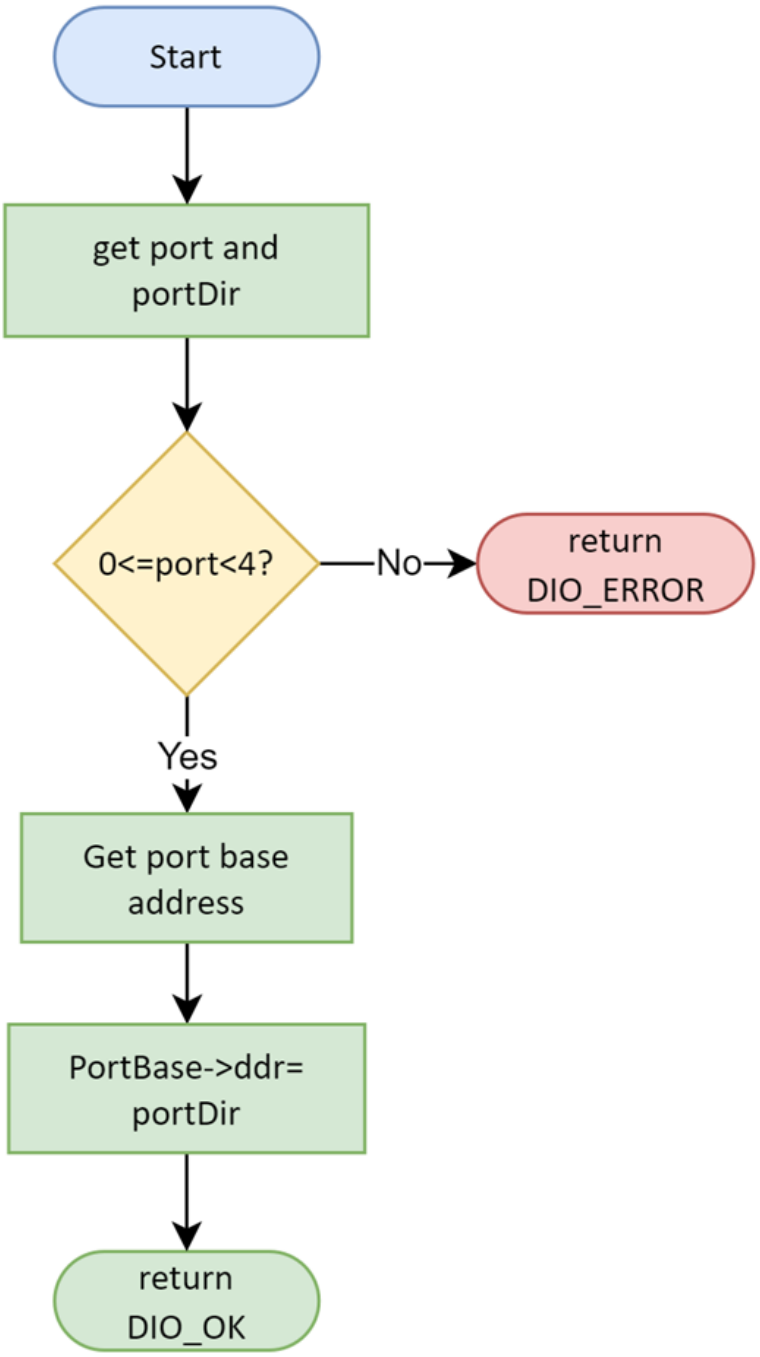
3.1.1.3. DIO_getPinVal



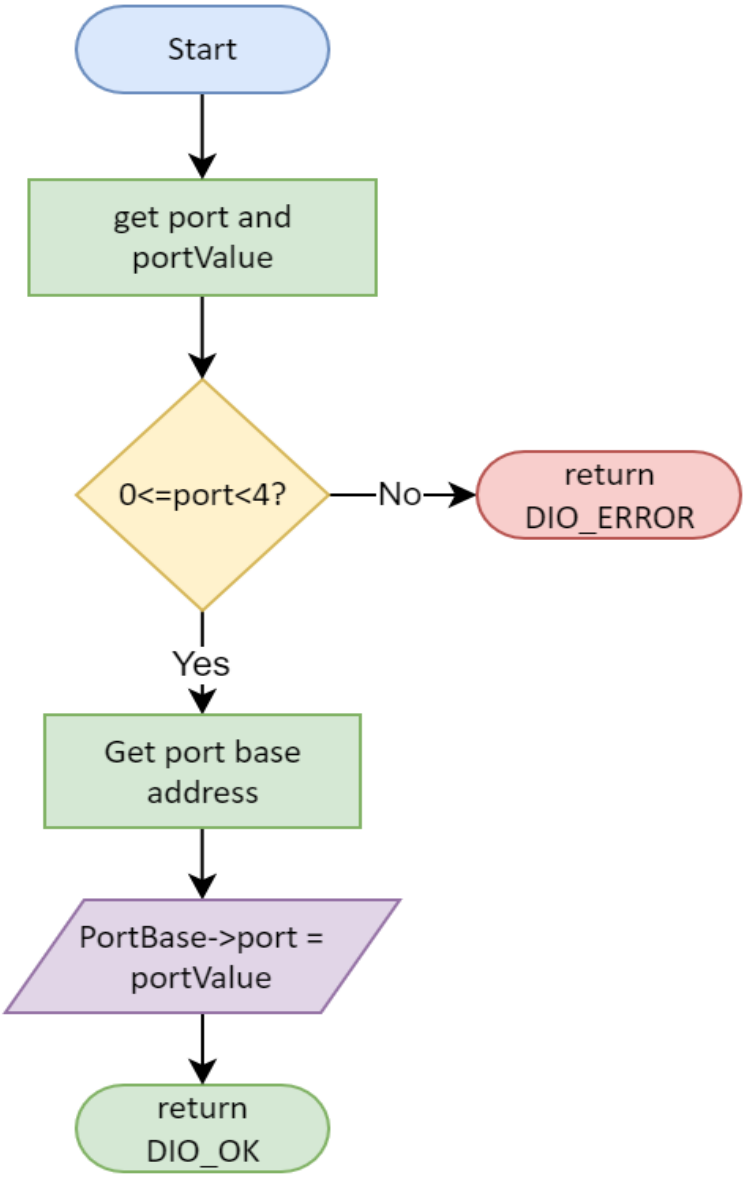
3.1.1.4. DIO_togPinVal



3.1.1.5. DIO_setPortDir

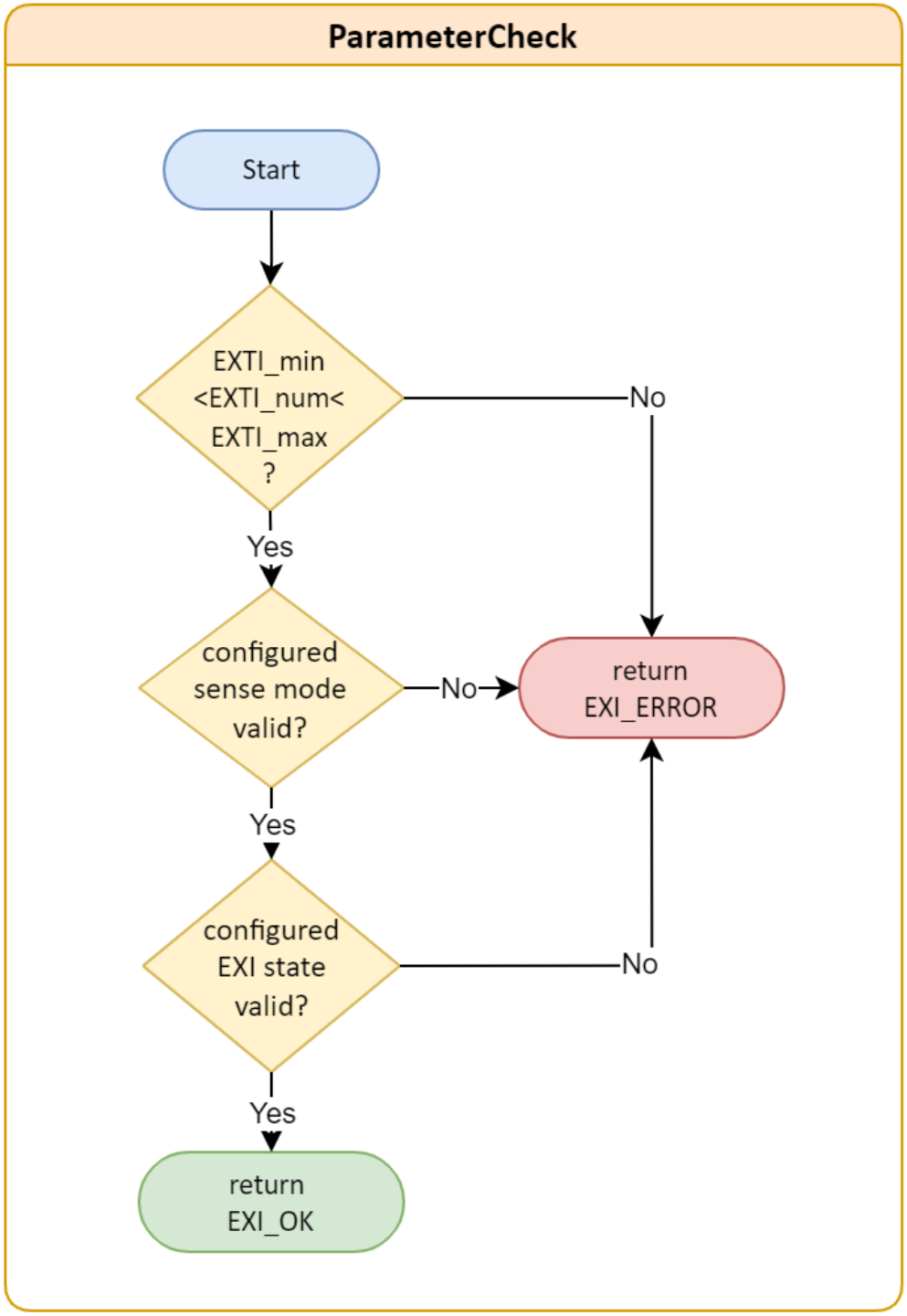


3.1.1.6. DIO_setPortVal

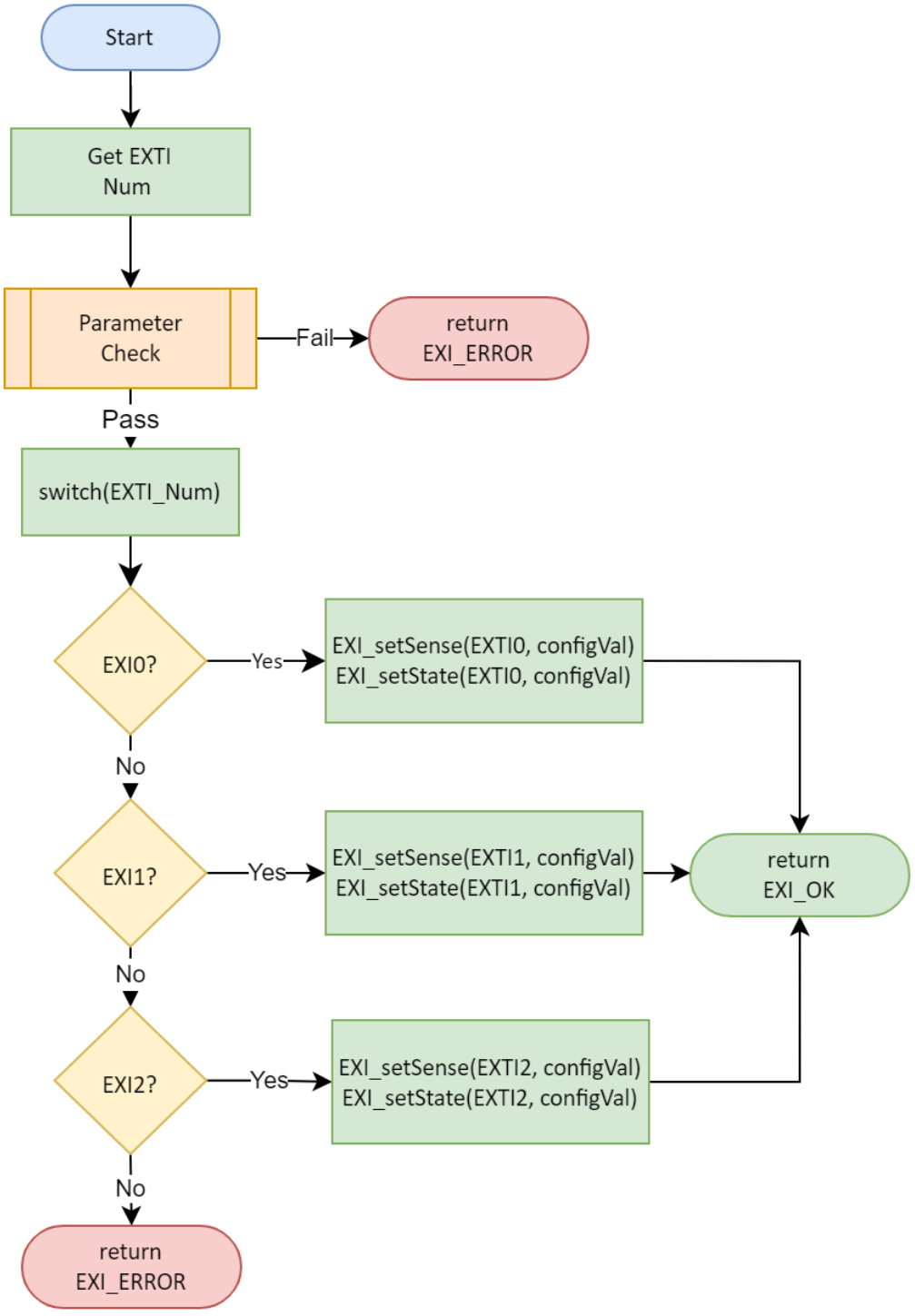


3.1.2. EXI Module

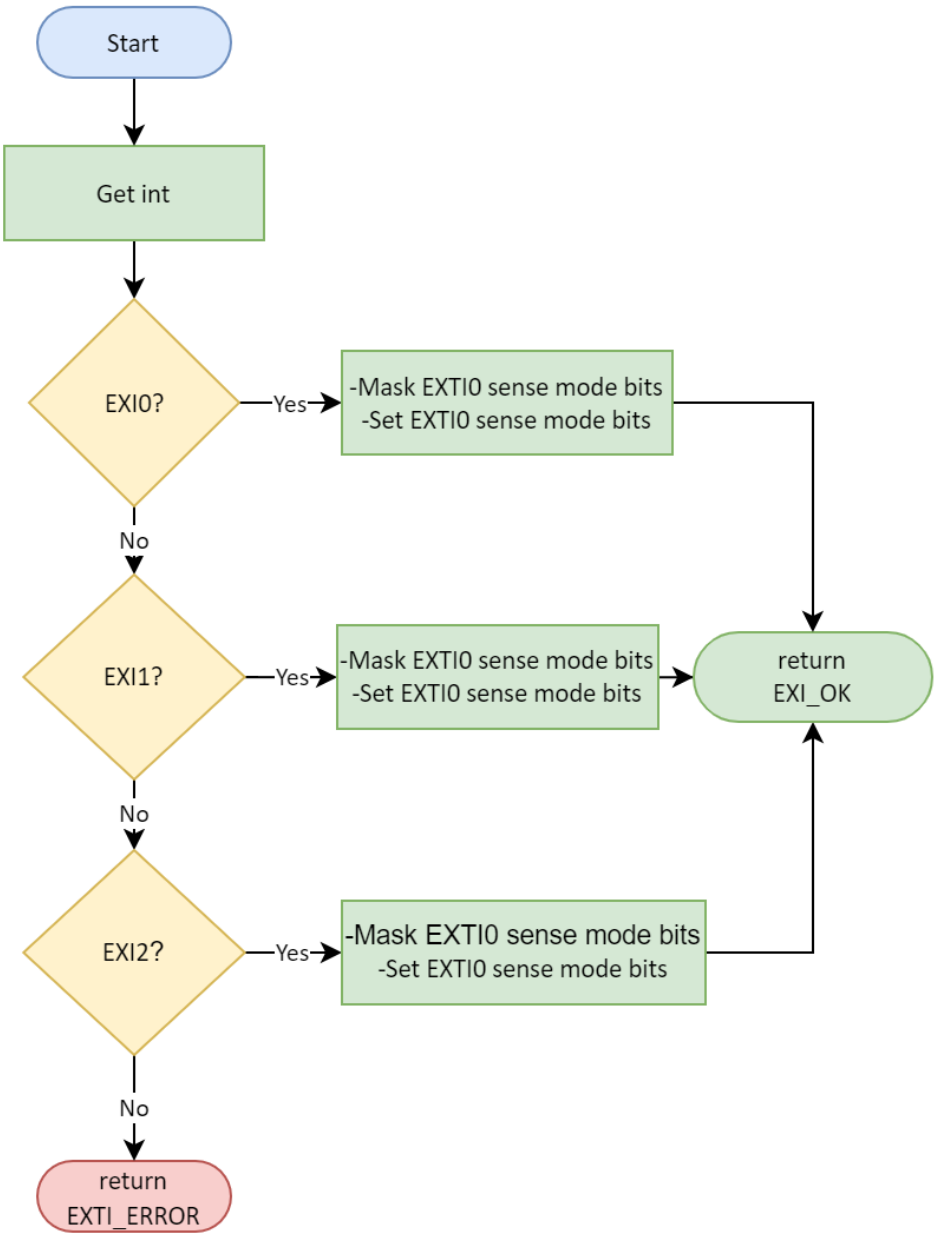
3.1.2.a. Sub process



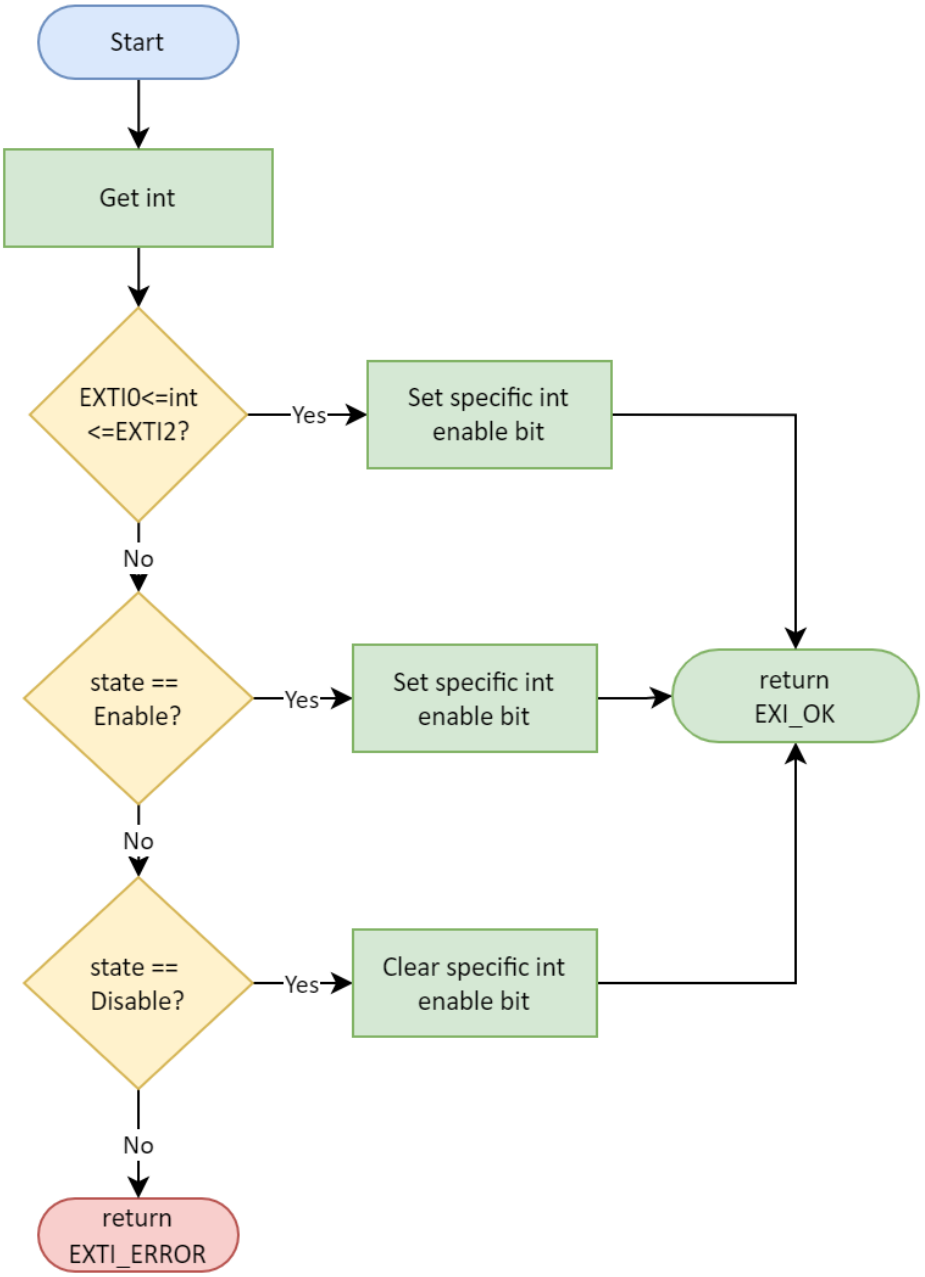
3.1.2.1. EXI_init



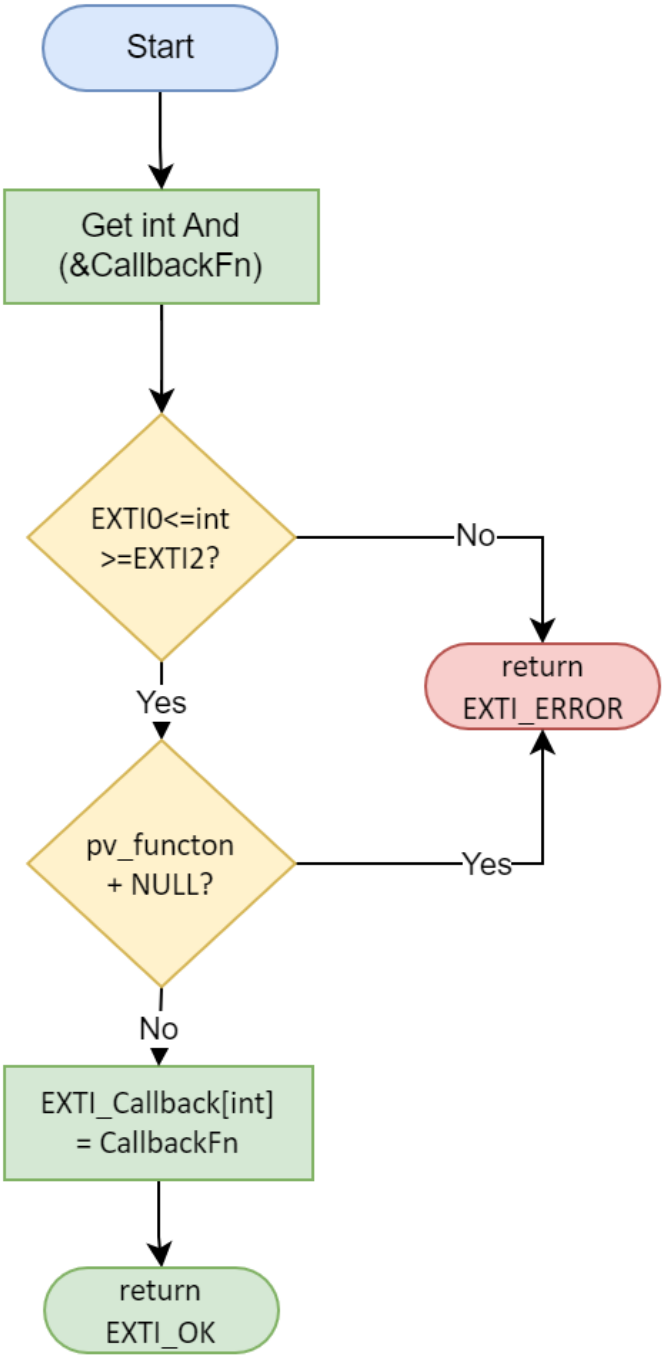
3.1.2.2. EXI_setSense



3.1.2.3. EXI_setState

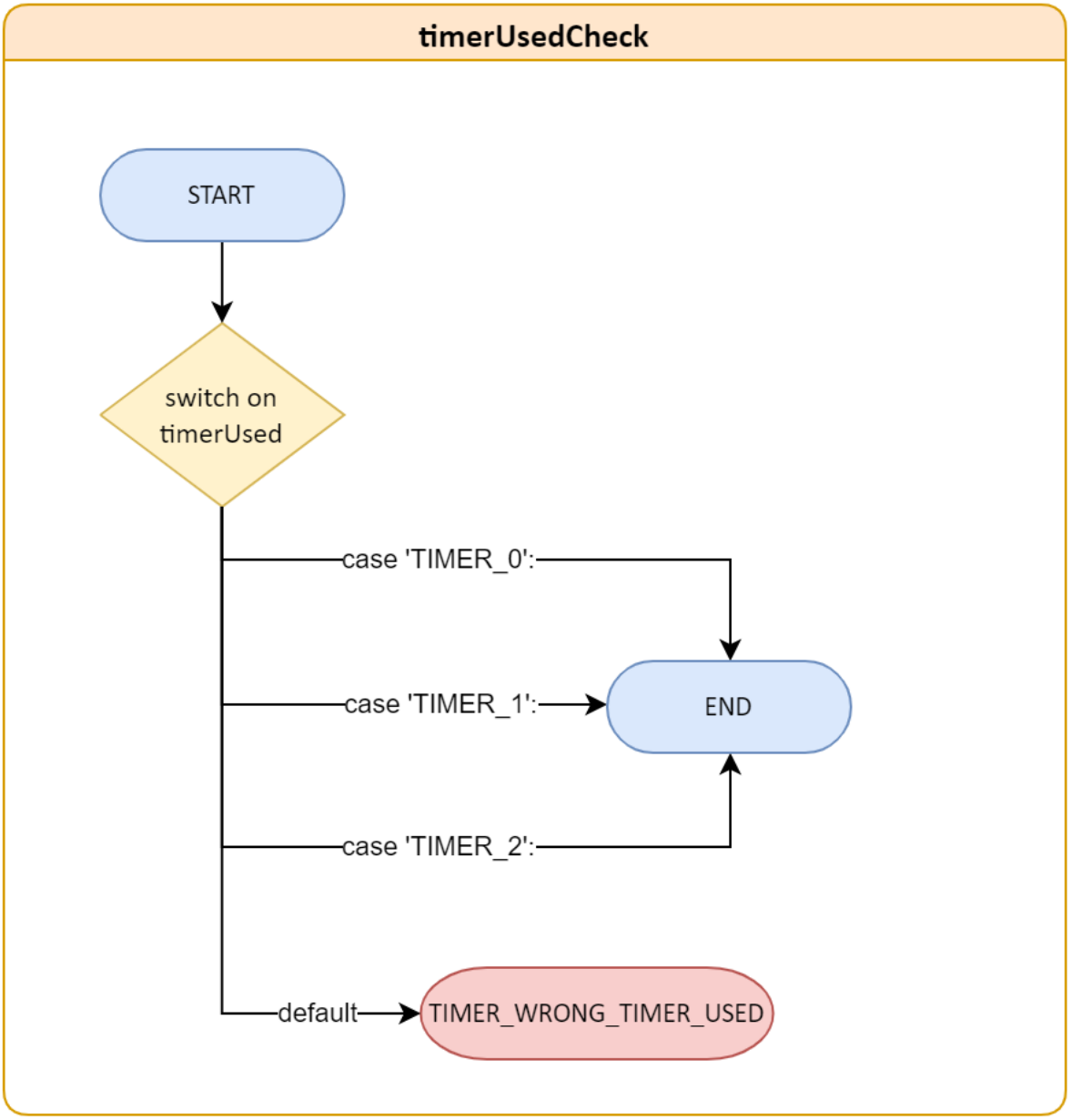


3.1.2.4. EXI_setCallback

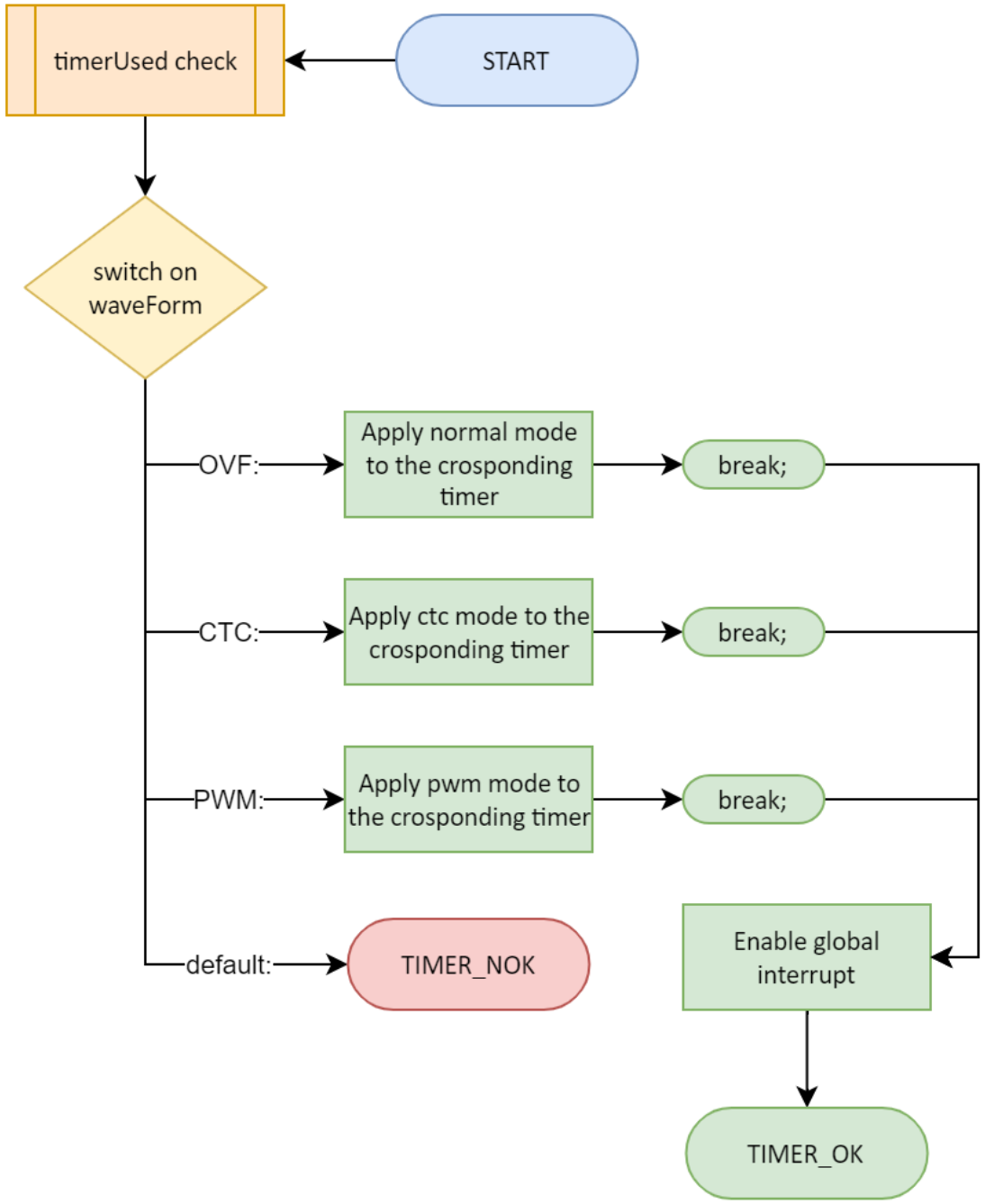


3.1.3. Timer Module

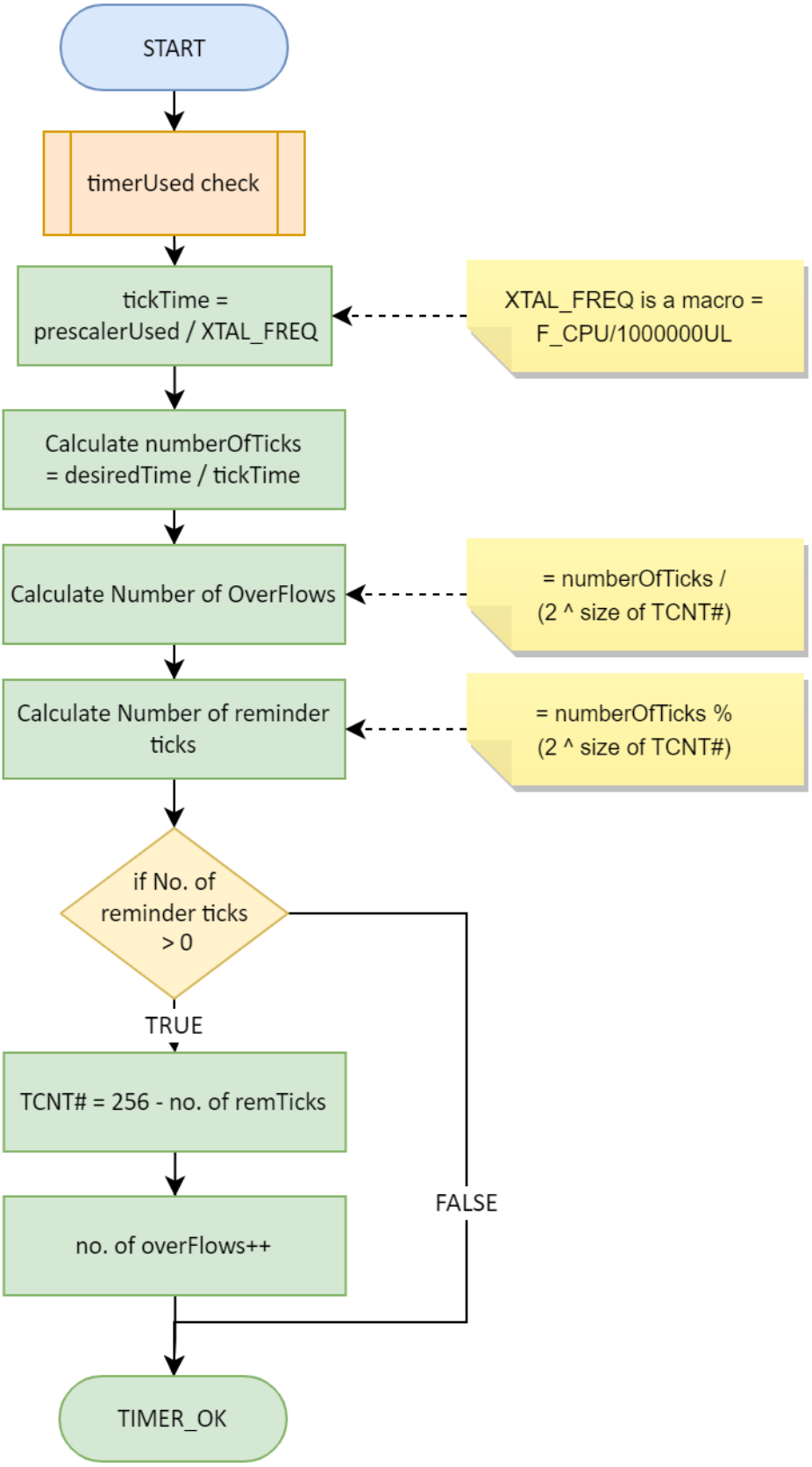
3.1.3.a. sub process



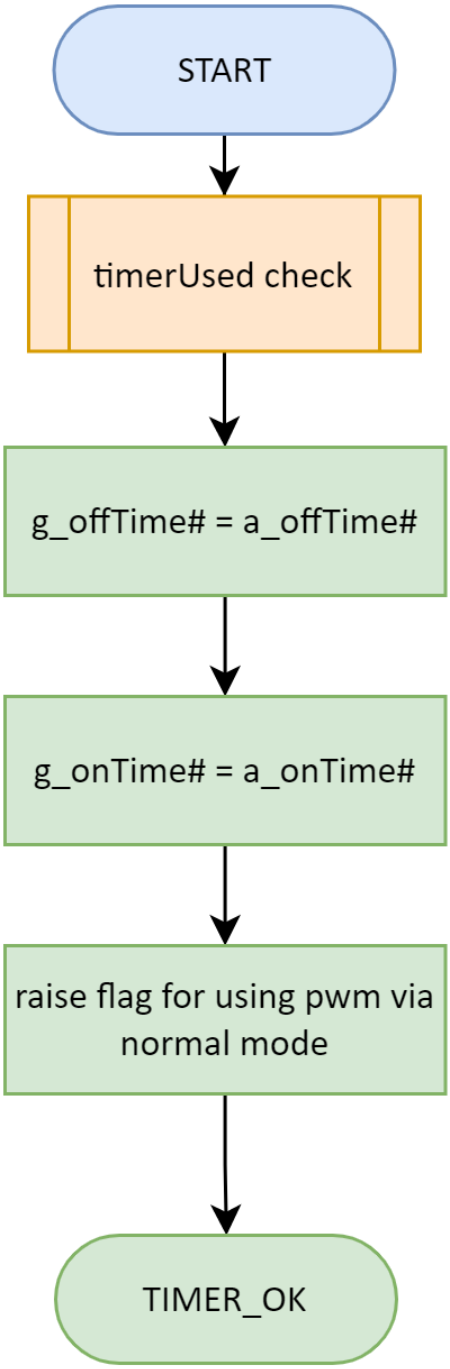
3.1.3.1. TIMER_init



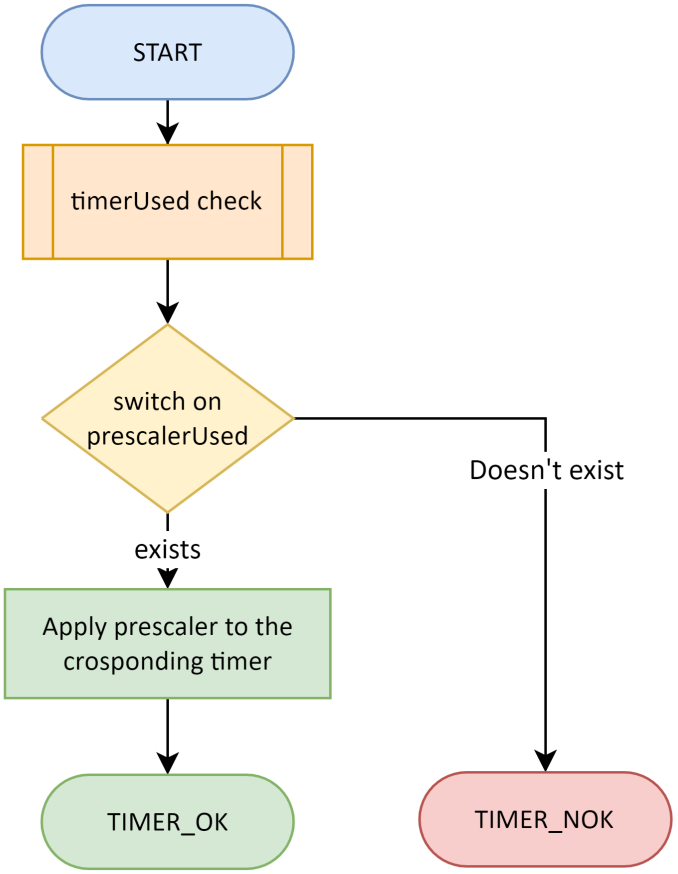
3.1.3.2. TIMER_setTime



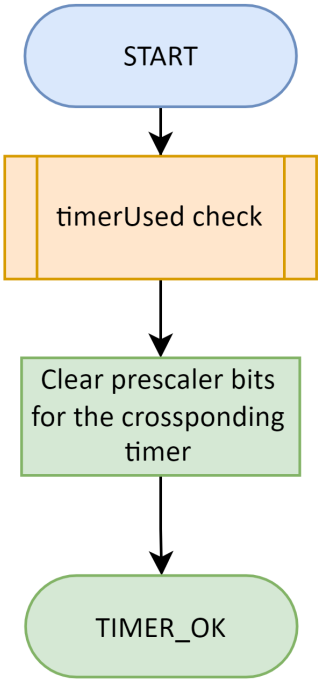
3.1.3.3. TIMER_pwmGenerator



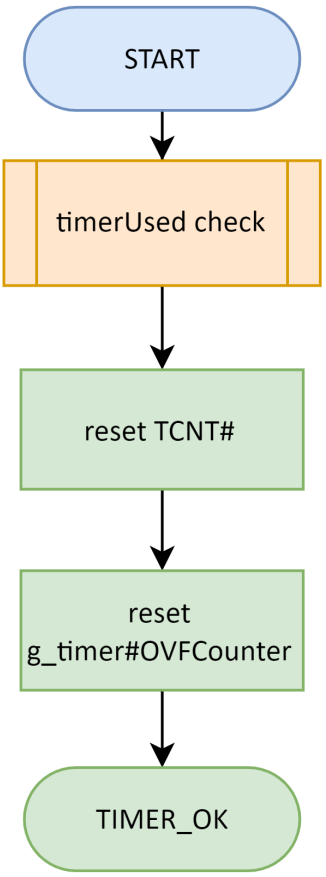
3.1.3.4. TIMER_resume



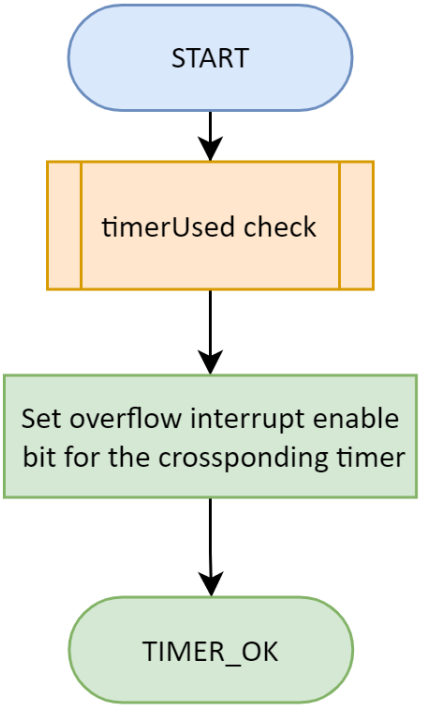
3.1.3.5. TIMER_pause



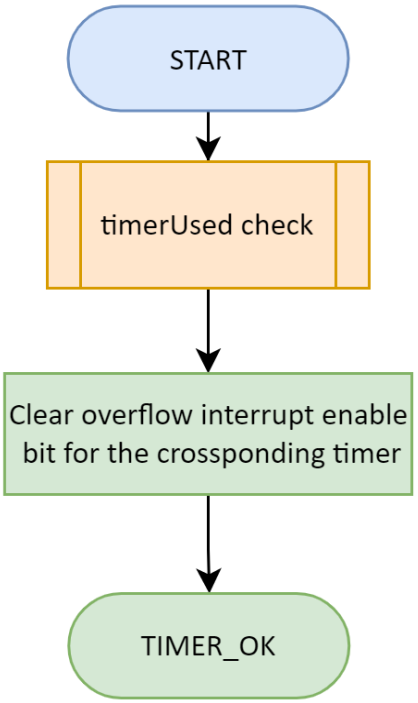
3.1.3.6. TIMER_reset



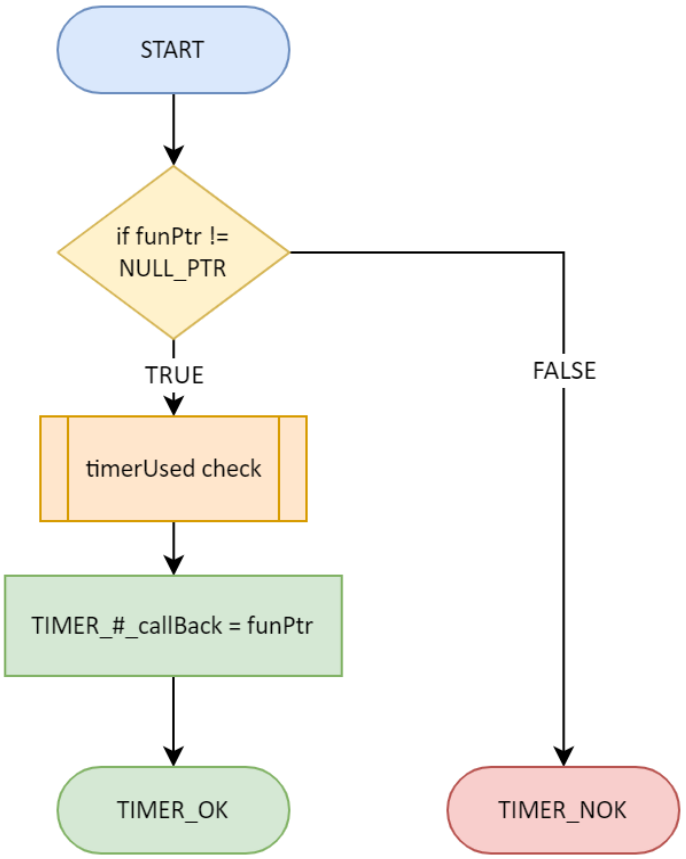
3.1.3.7. TIMER_enableInterrupt



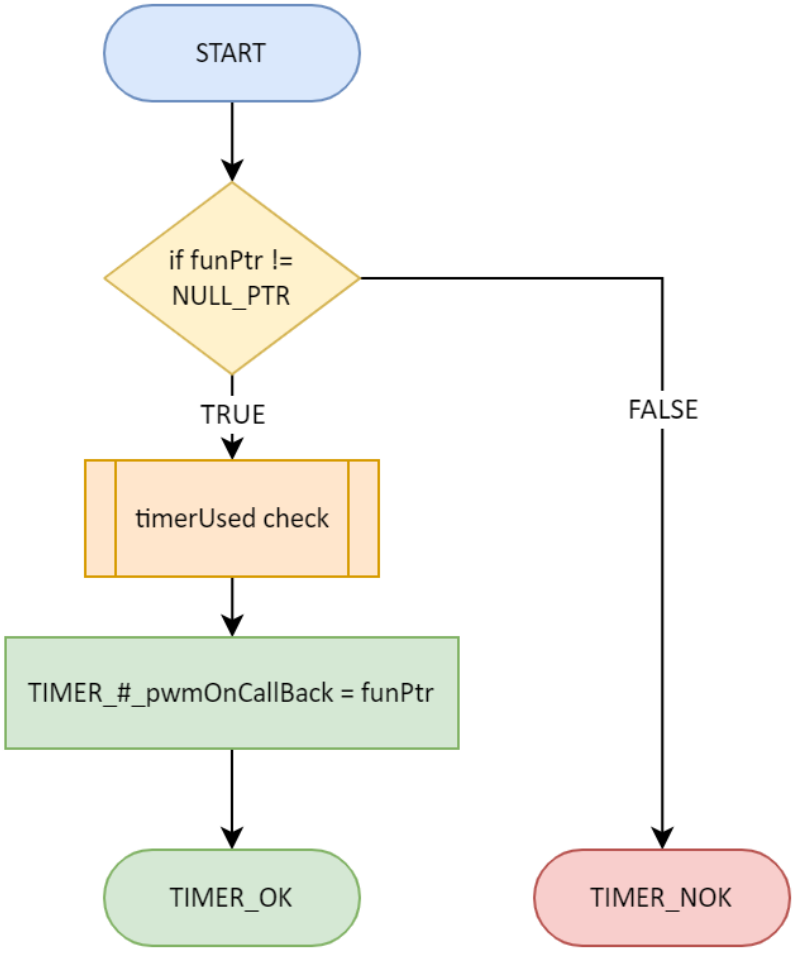
3.1.3.8. TIMER_enableInterrupt



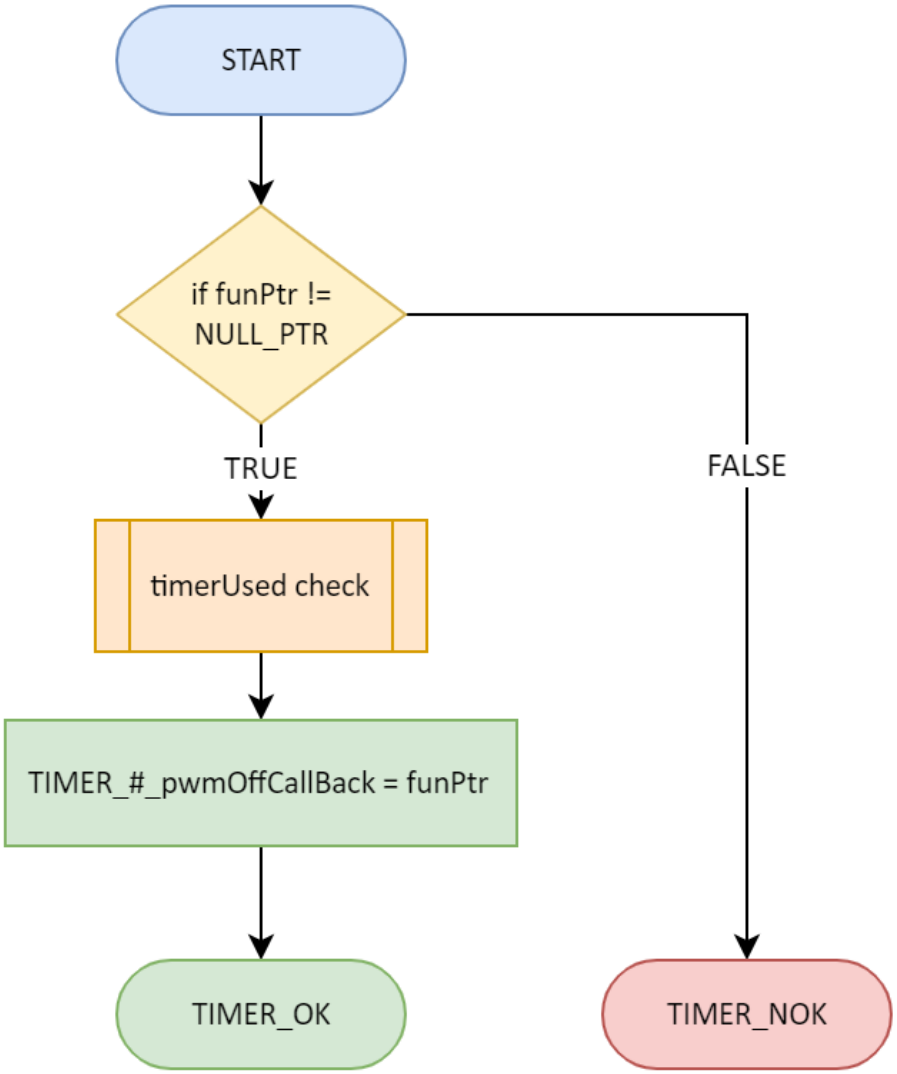
3.1.3.9. TIMER_setCallBack



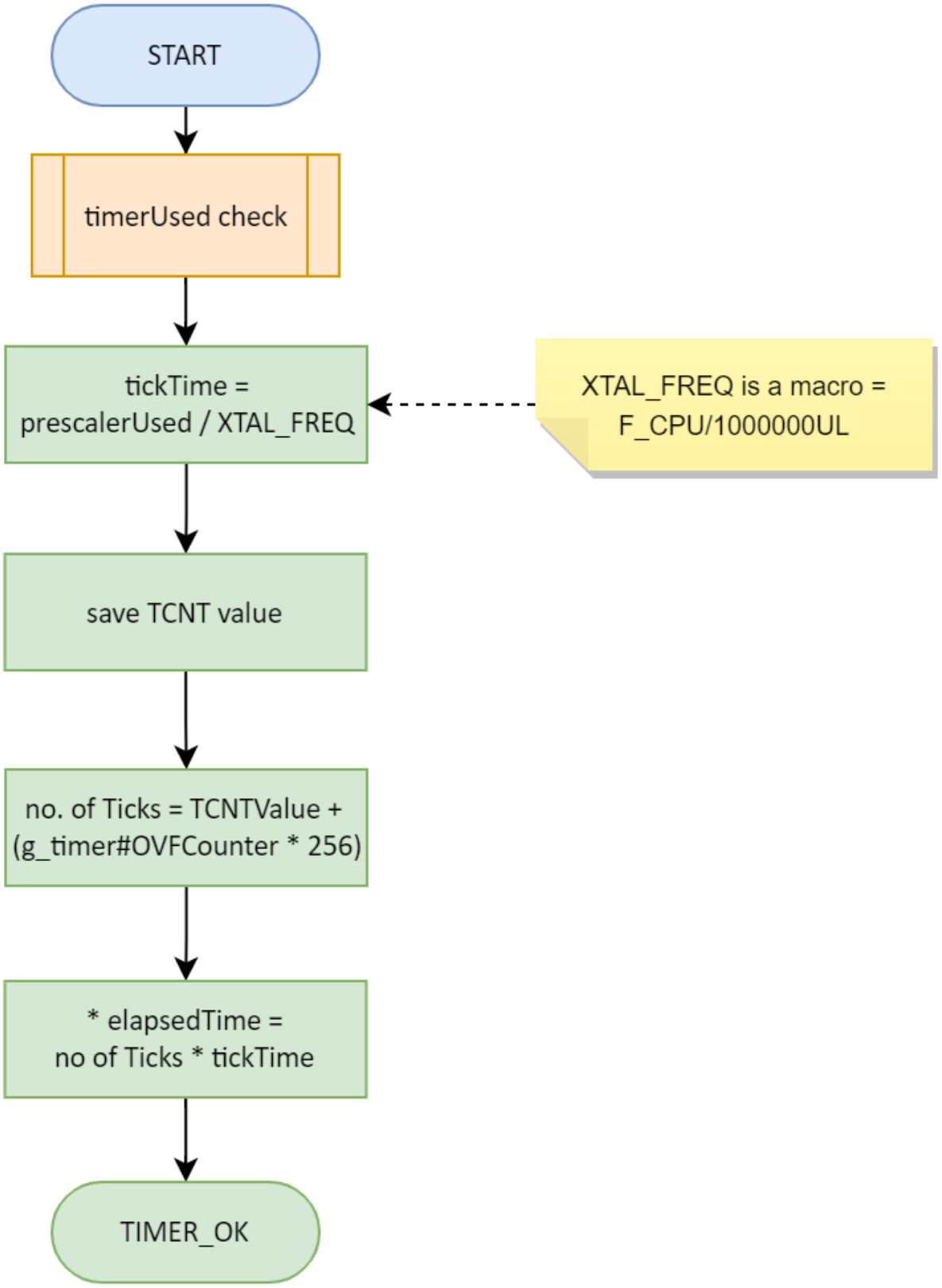
3.1.3.10. TIMER_setPwmOnCallBack



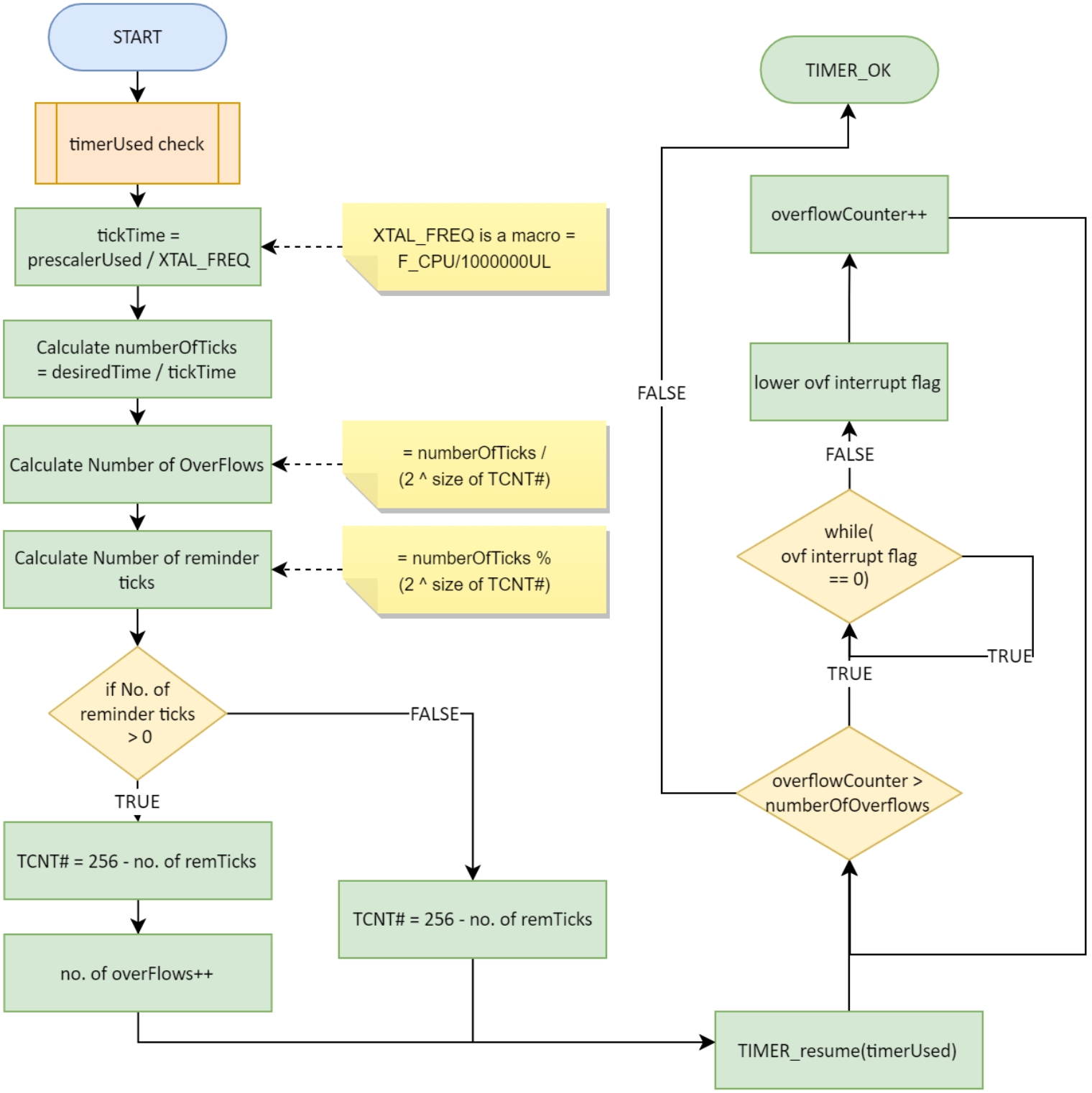
3.1.3.11. TIMER_setPwmOffCallBack



3.1.3.12. TIMER_getElapsedTime

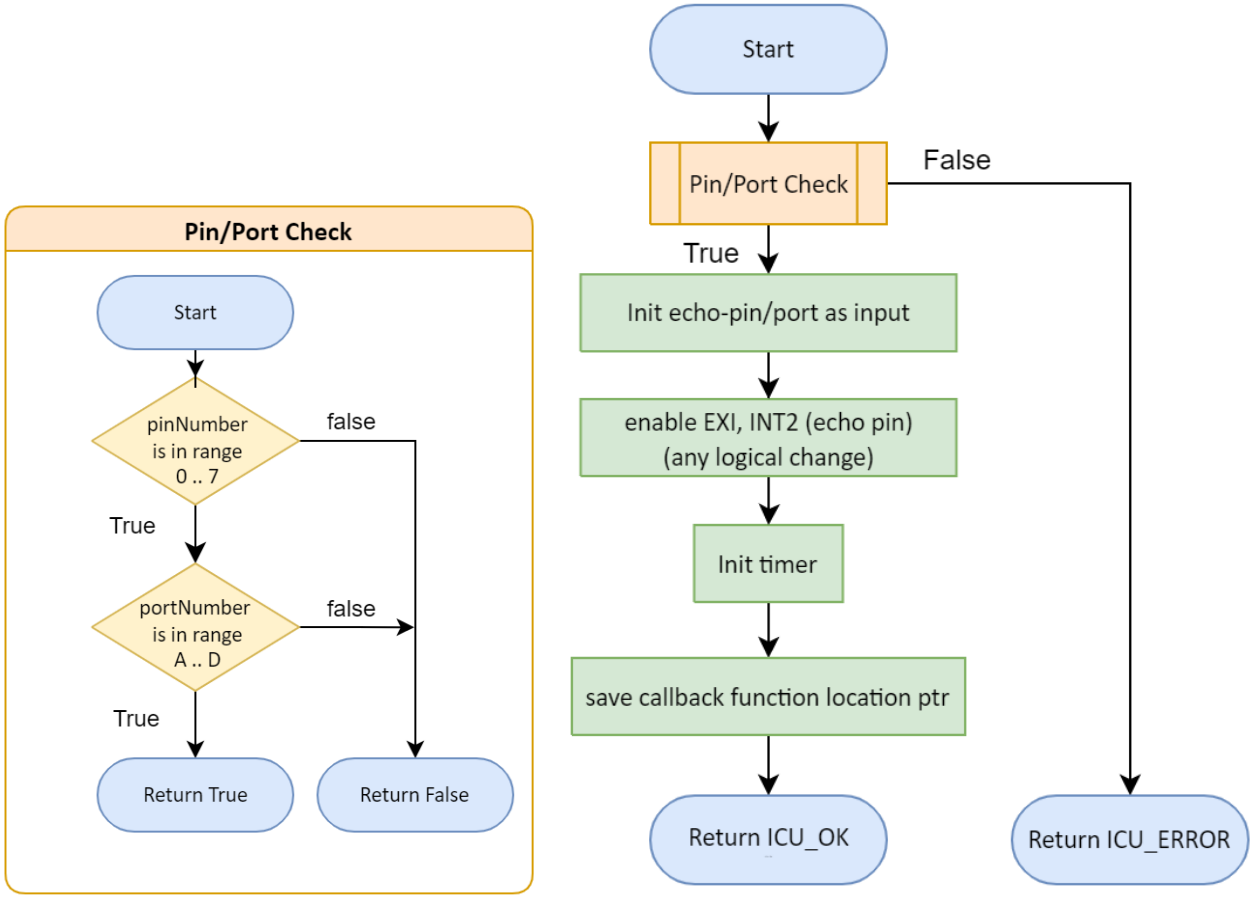


3.1.3.13. TIMER_setDelayTime

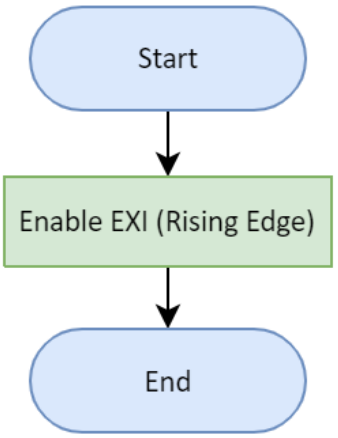


3.1.4. ICU Module (Input Capture Unit)

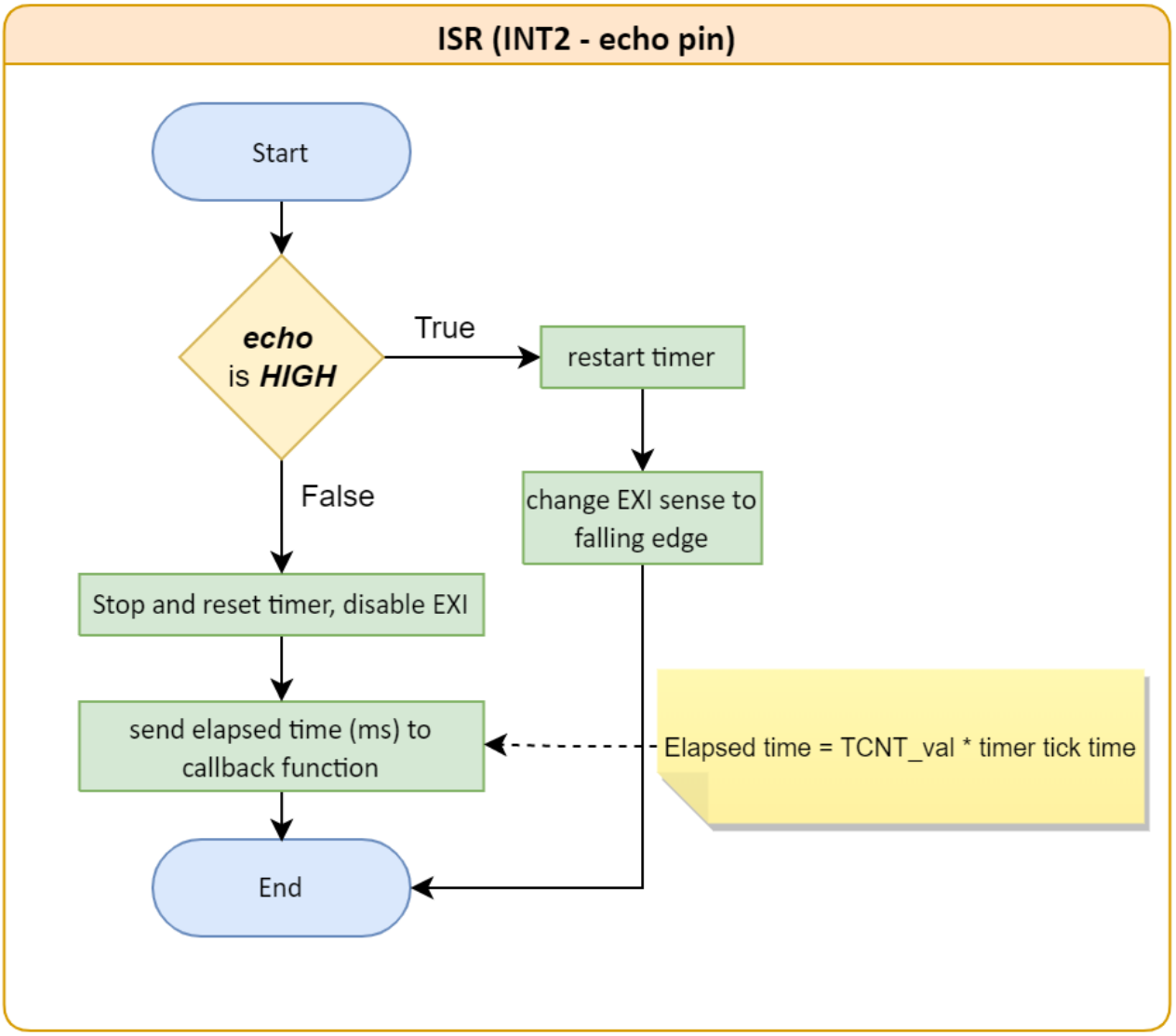
3.1.4.1. ICU_init



3.1.4.2. ICU_getCaptureValue



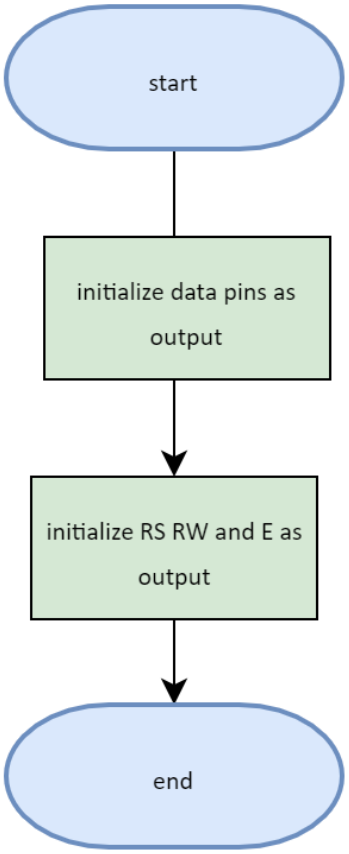
3.1.4.3. ISR (INT2 - Echo pin)



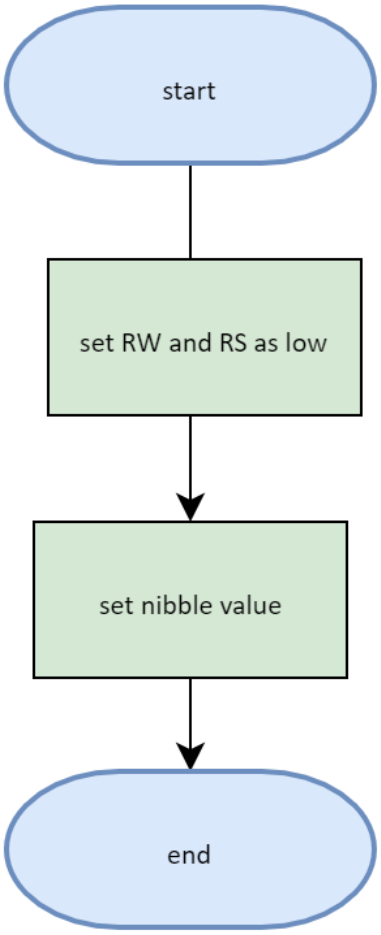
3.2. HAL Layer

3.2.1. LCD Module

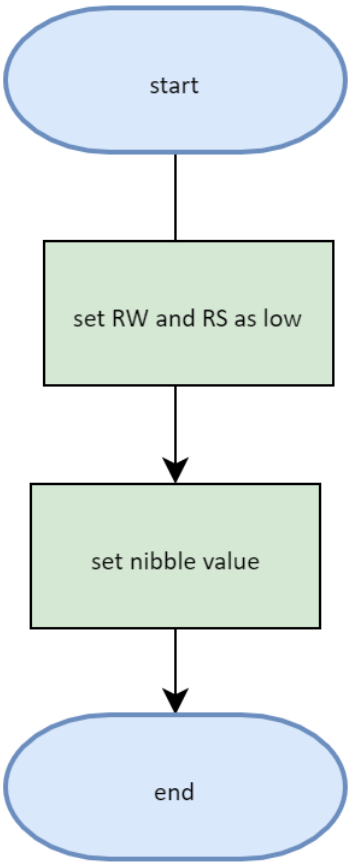
3.2.1.1. LCD_init



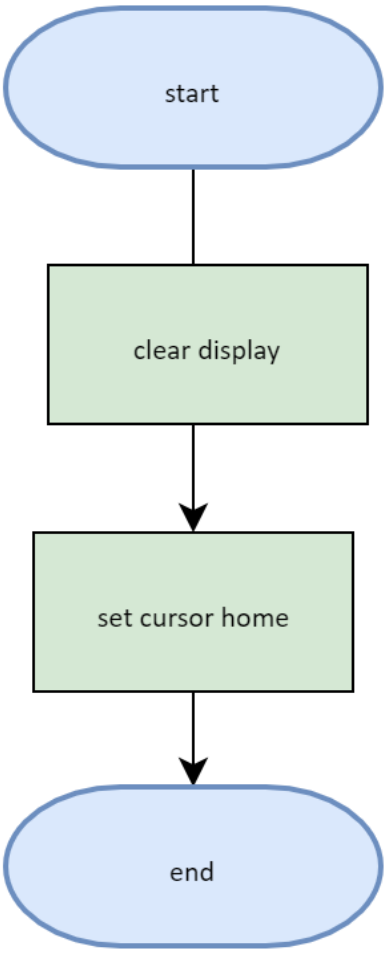
3.2.1.2. LCD_writecmd



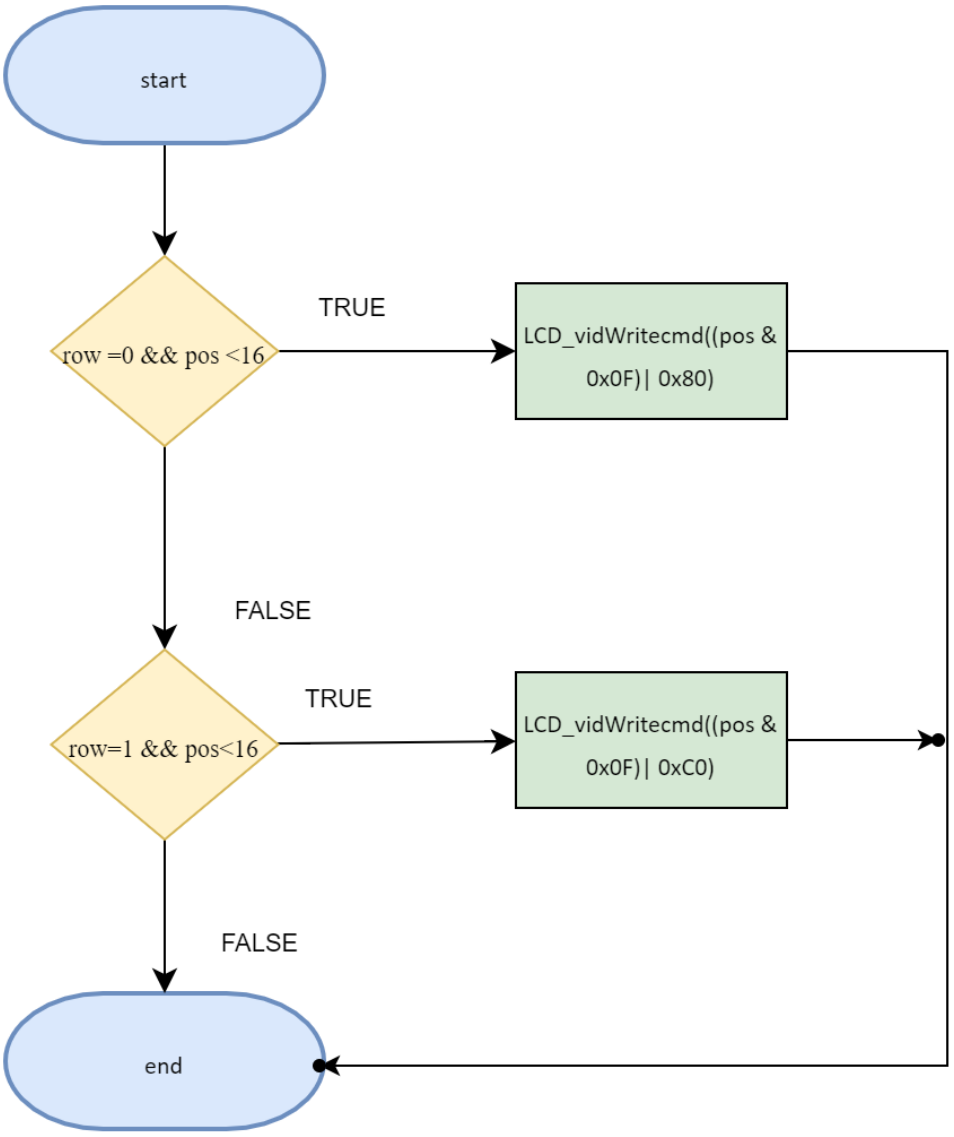
3.2.1.3. LCD_writeChar



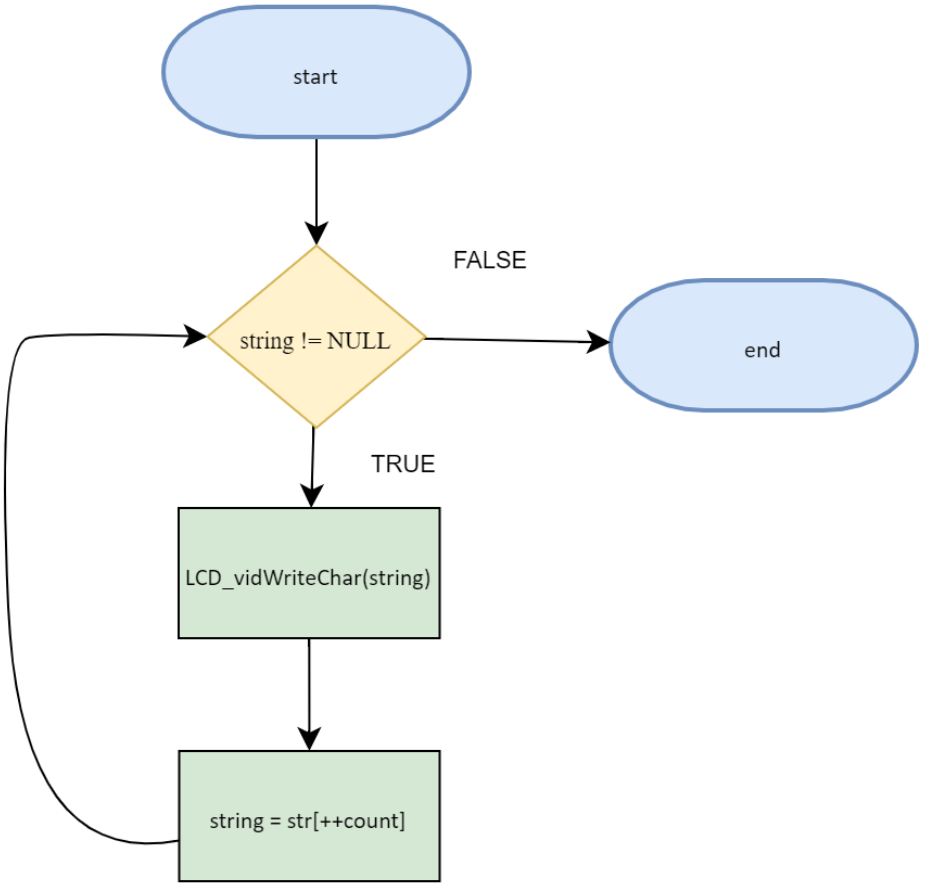
3.2.1.4. LCD_clrDisplay



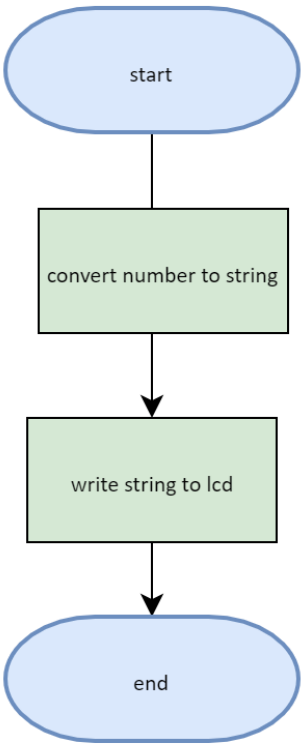
3.2.1.5. LCD_gotoXY



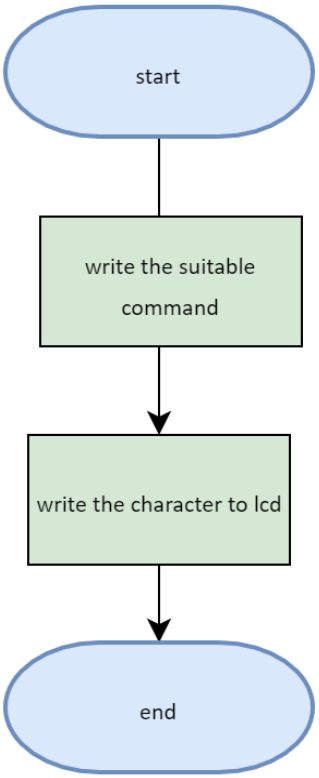
3.2.1.6. LCD_writeString



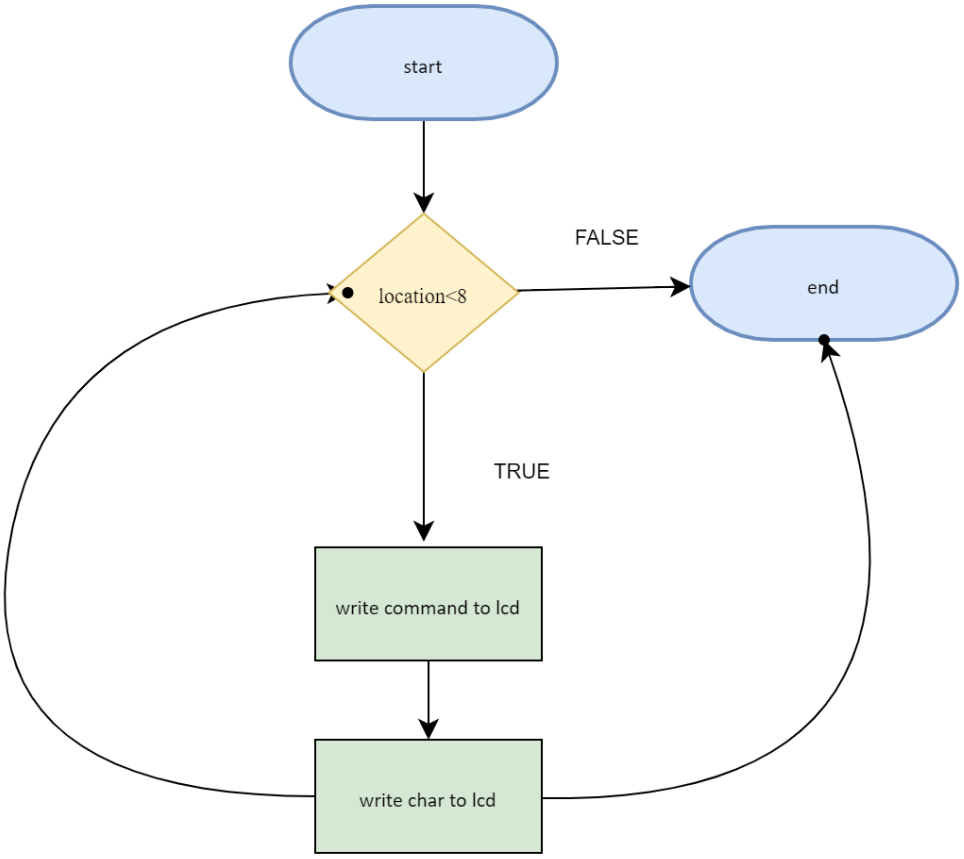
3.2.1.7. LCD_writeInt



3.2.1.8. LCD_writeArabic

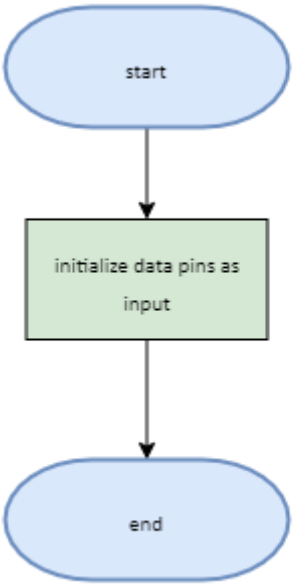


3.2.1.9. LCD_createCustomChar

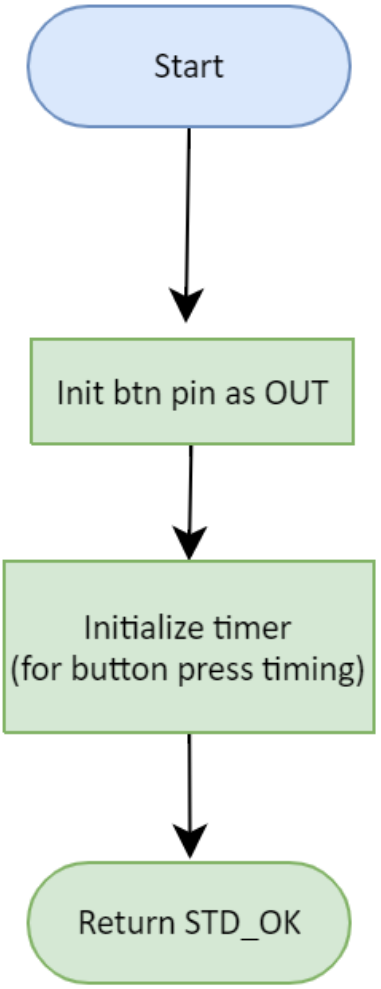


3.2.2. BTN Module

3.2.2.1. BUTTON_init

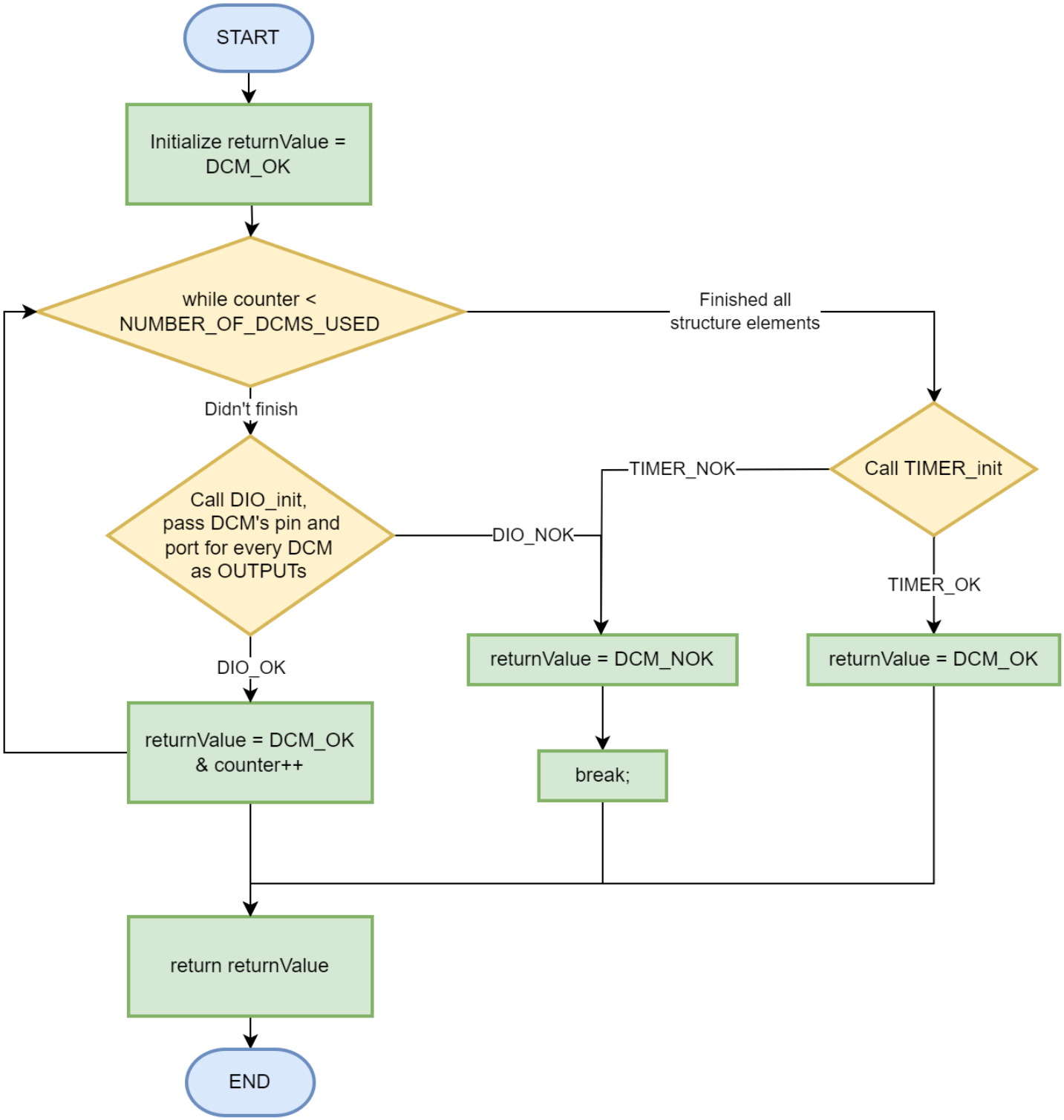


3.2.2.2. BUTTON_read

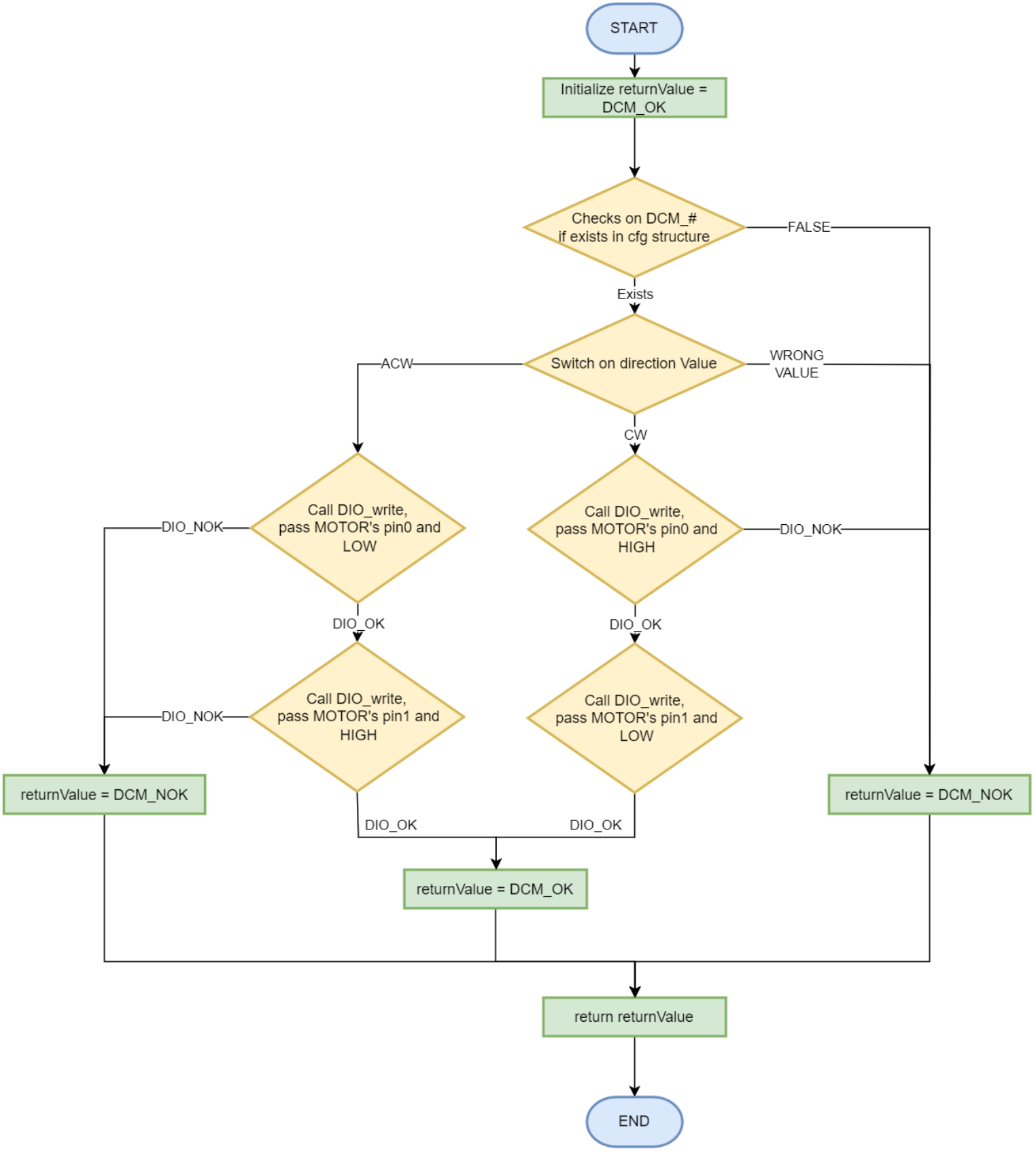


3.2.3. DCM Module

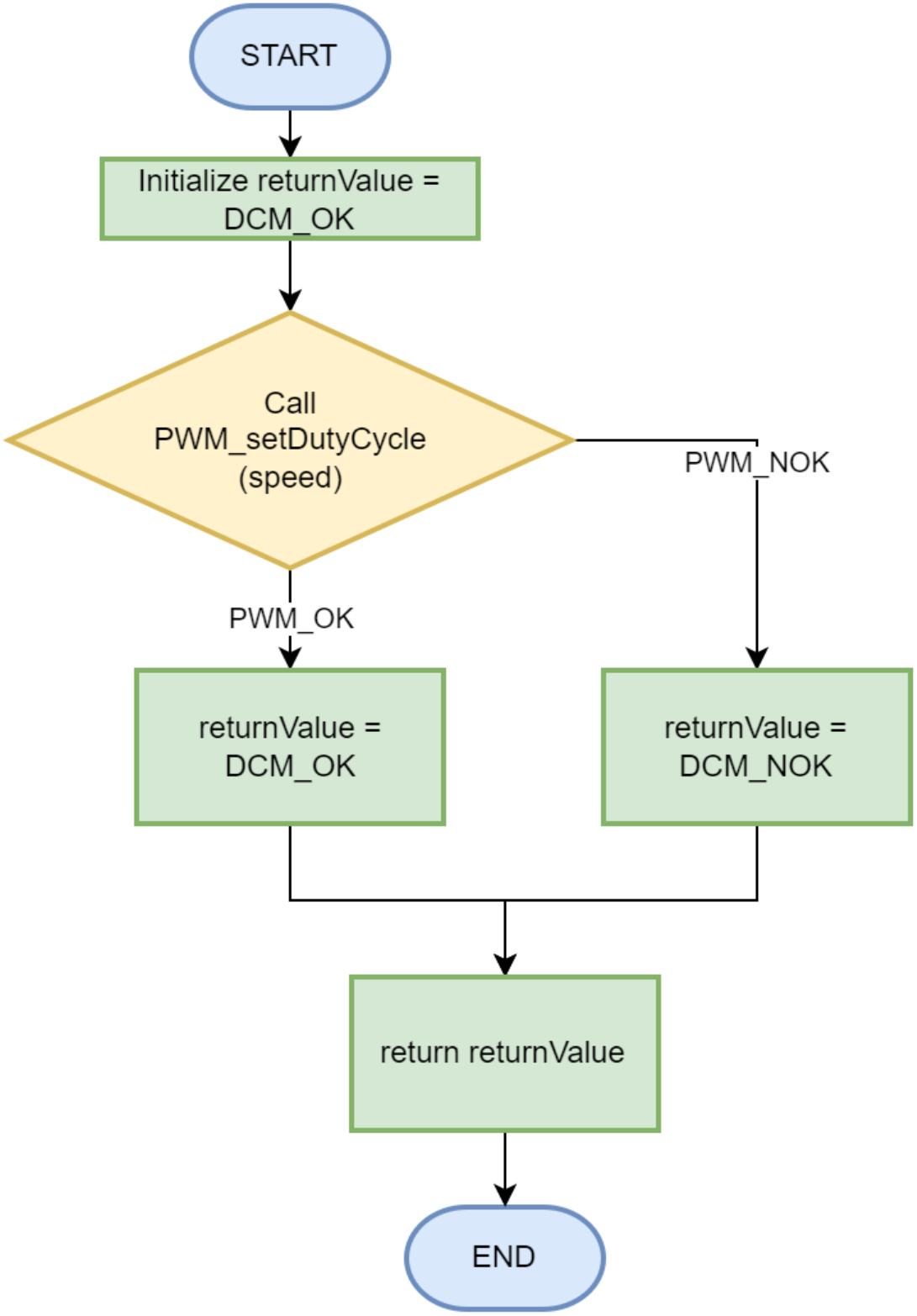
3.2.3.1. DCM_init



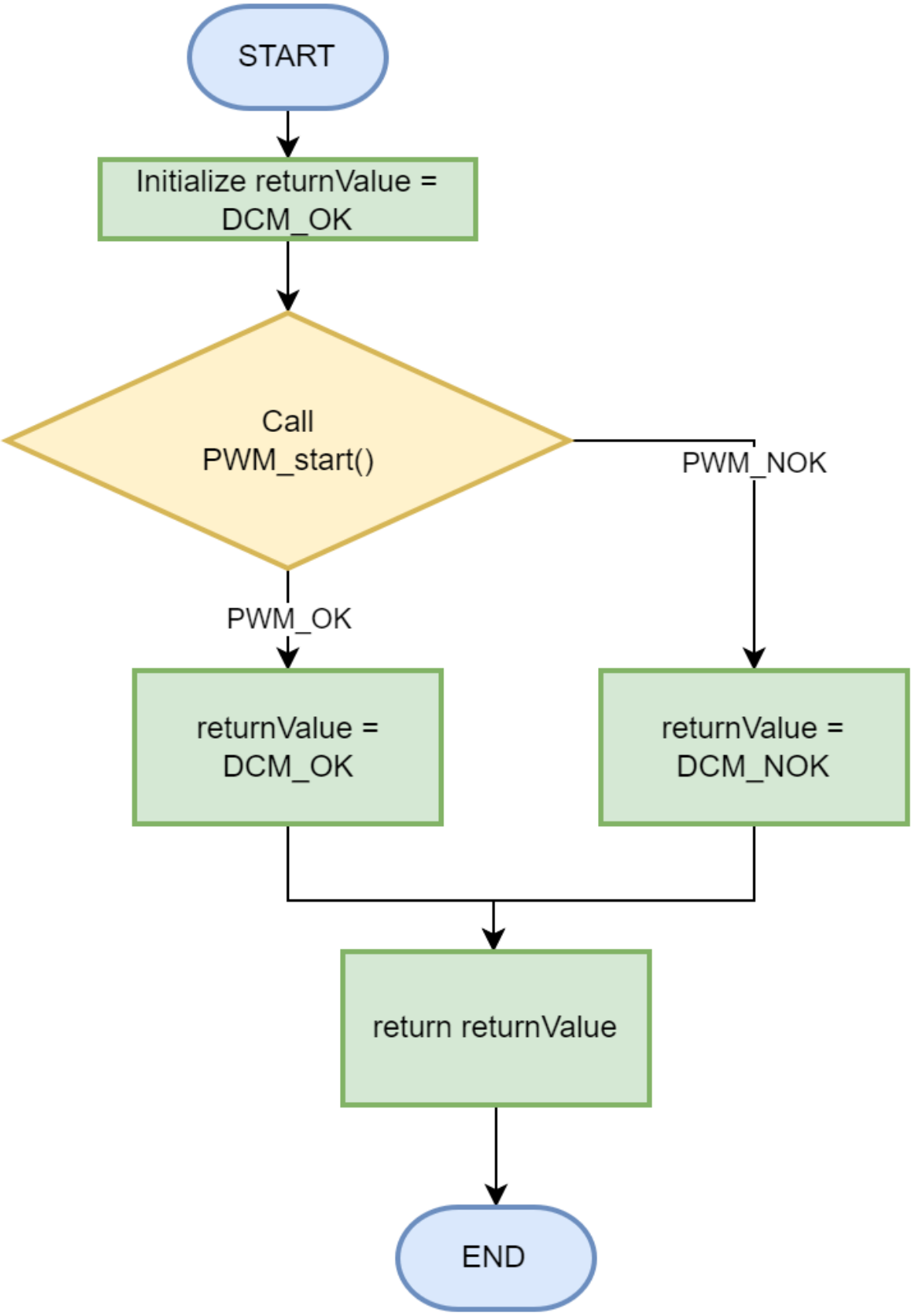
3.2.3.2. DCM_setDirection



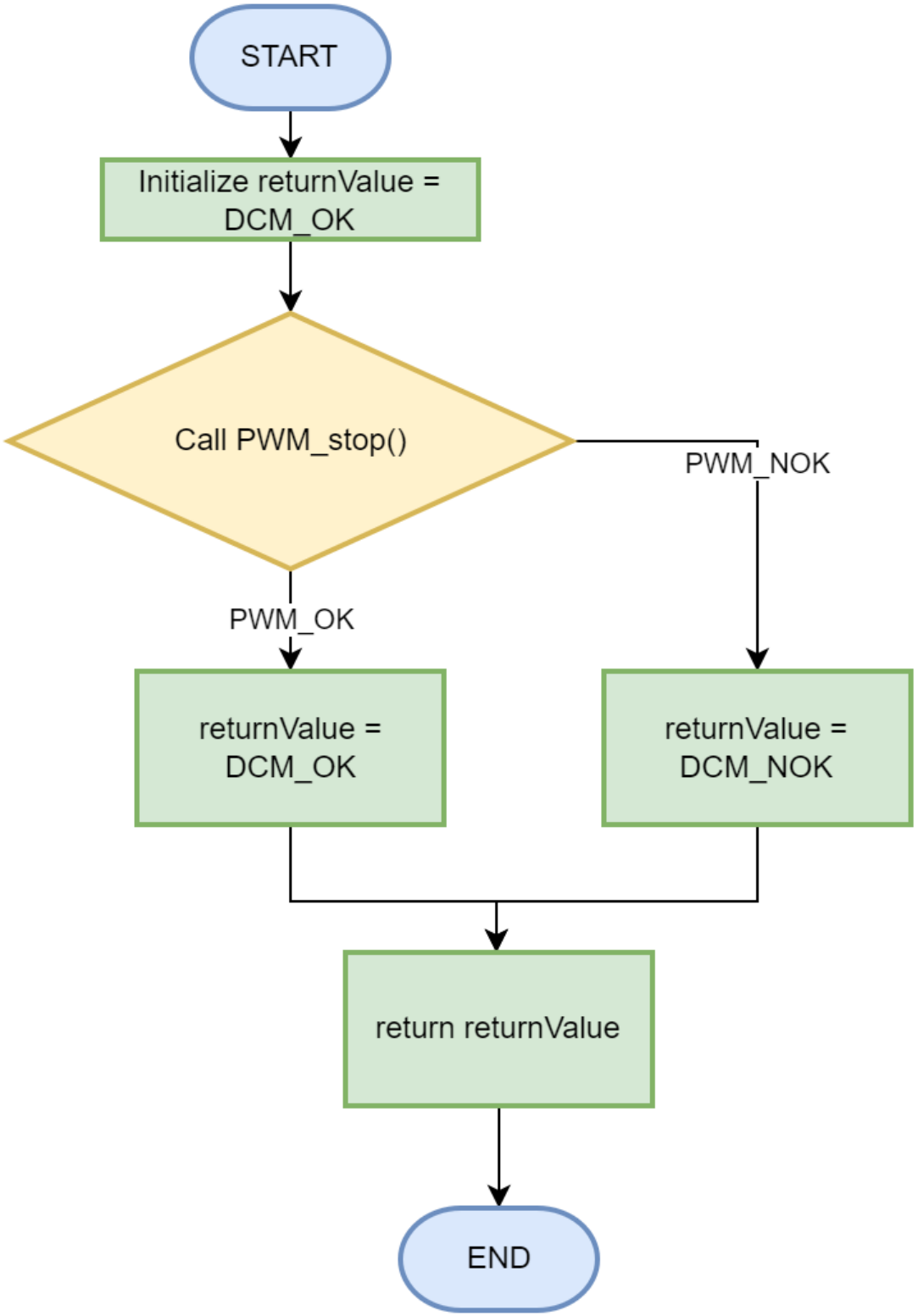
3.2.3.3. DCM_speed



3.2.3.4. DCM_start

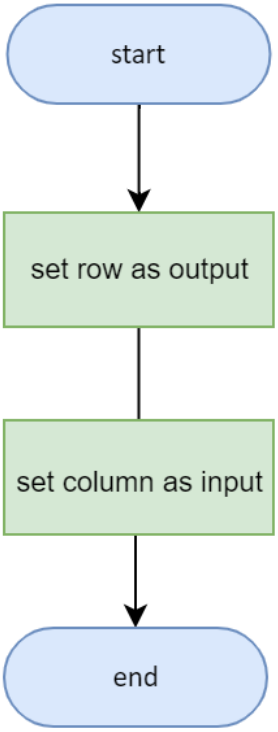


3.2.3.5. DCM_stop

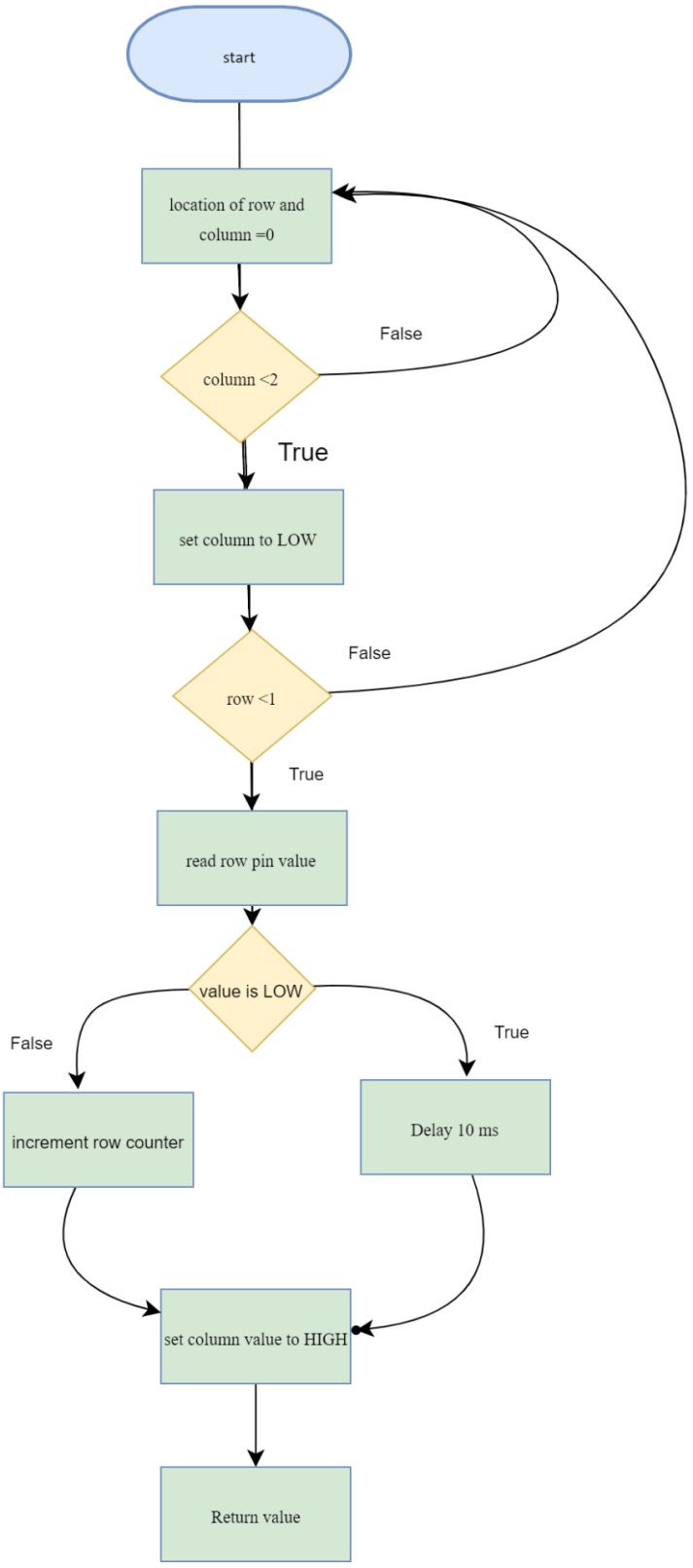


3.2.4. KPD Module

3.2.4.1. KEYPAD_init

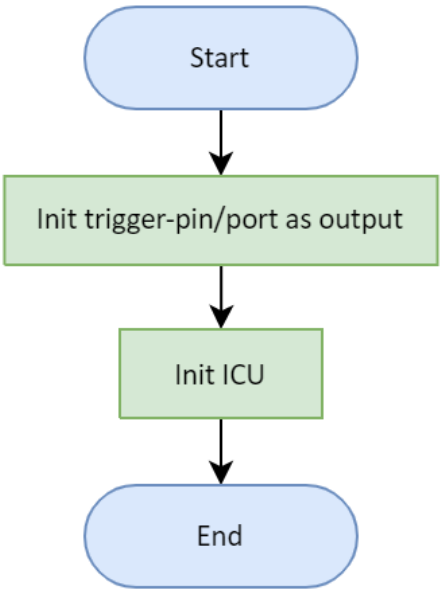


3.2.4.2. KEYPAD_getButton

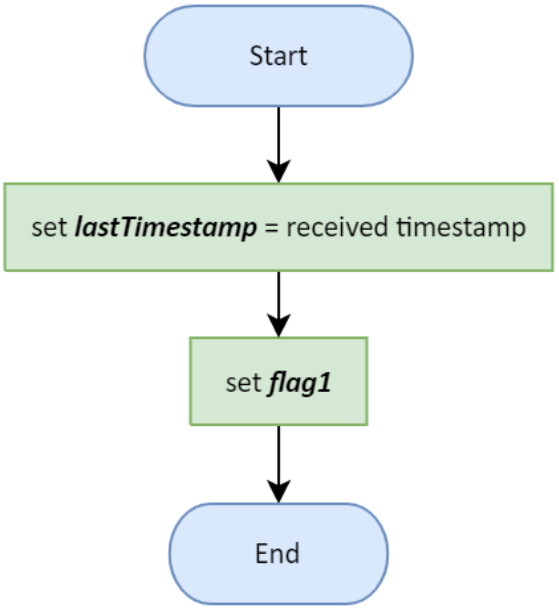


3.2.5. Ultrasonic Module

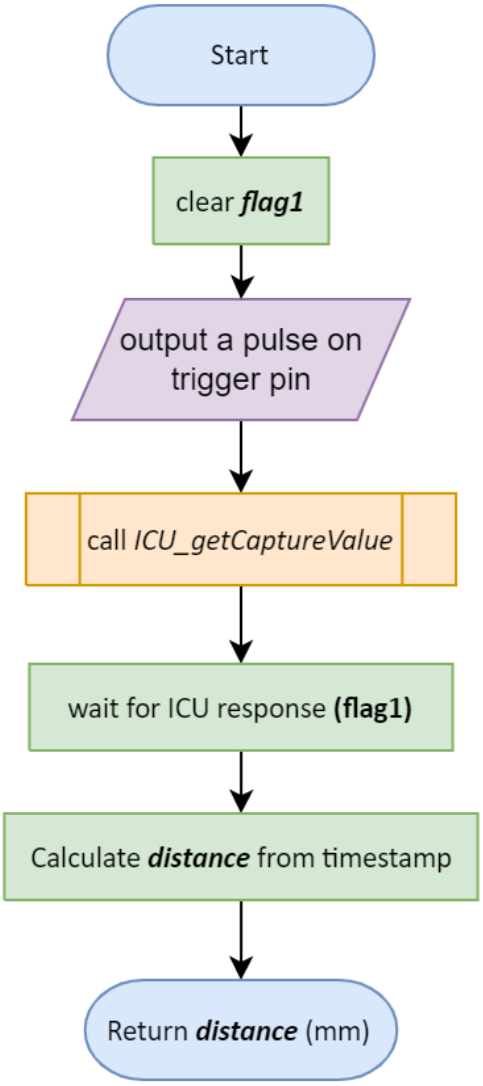
3.2.5.1. US_init



3.2.5.2. US_evtDistance

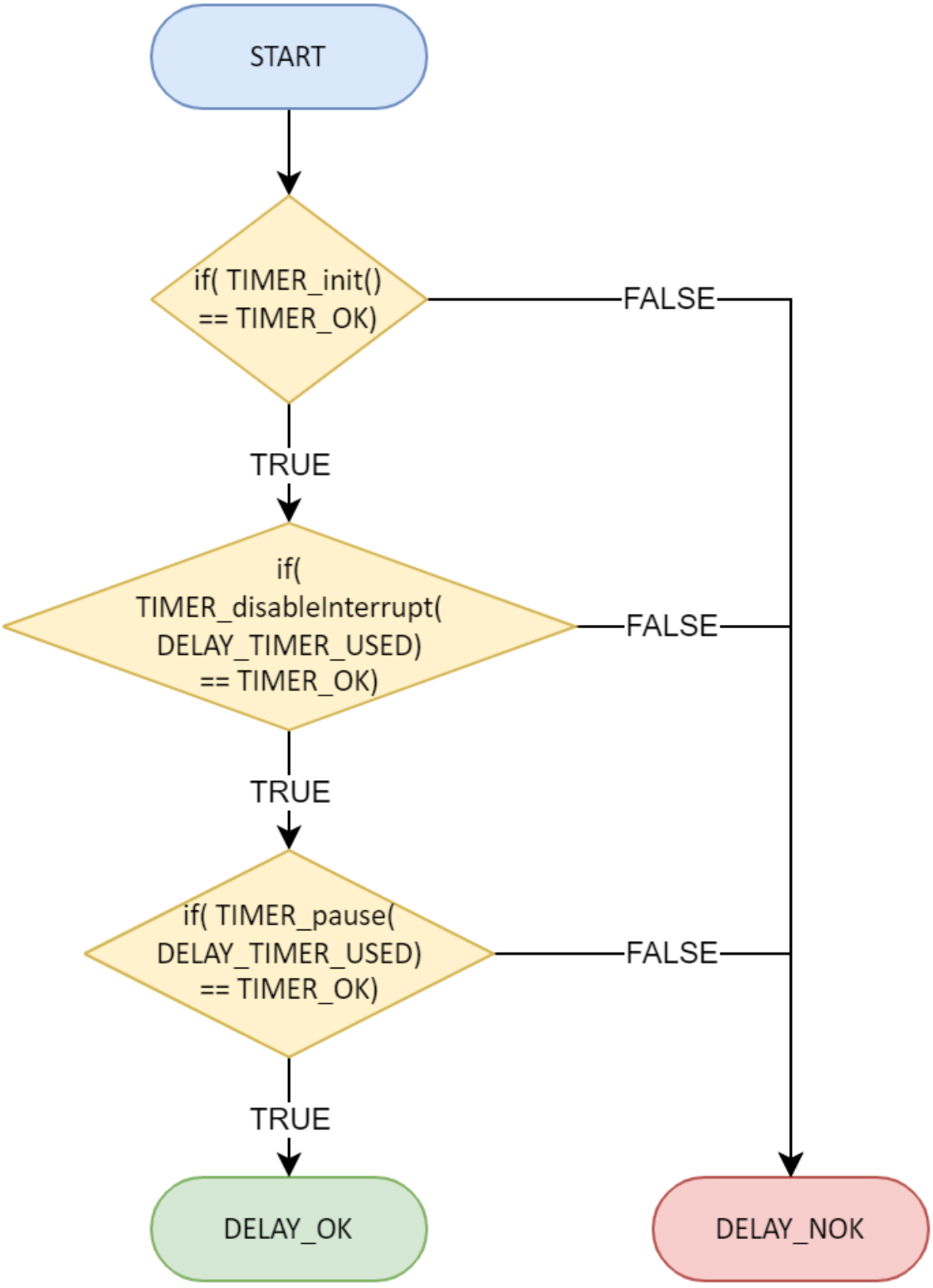


3.2.5.3. US_getDistance

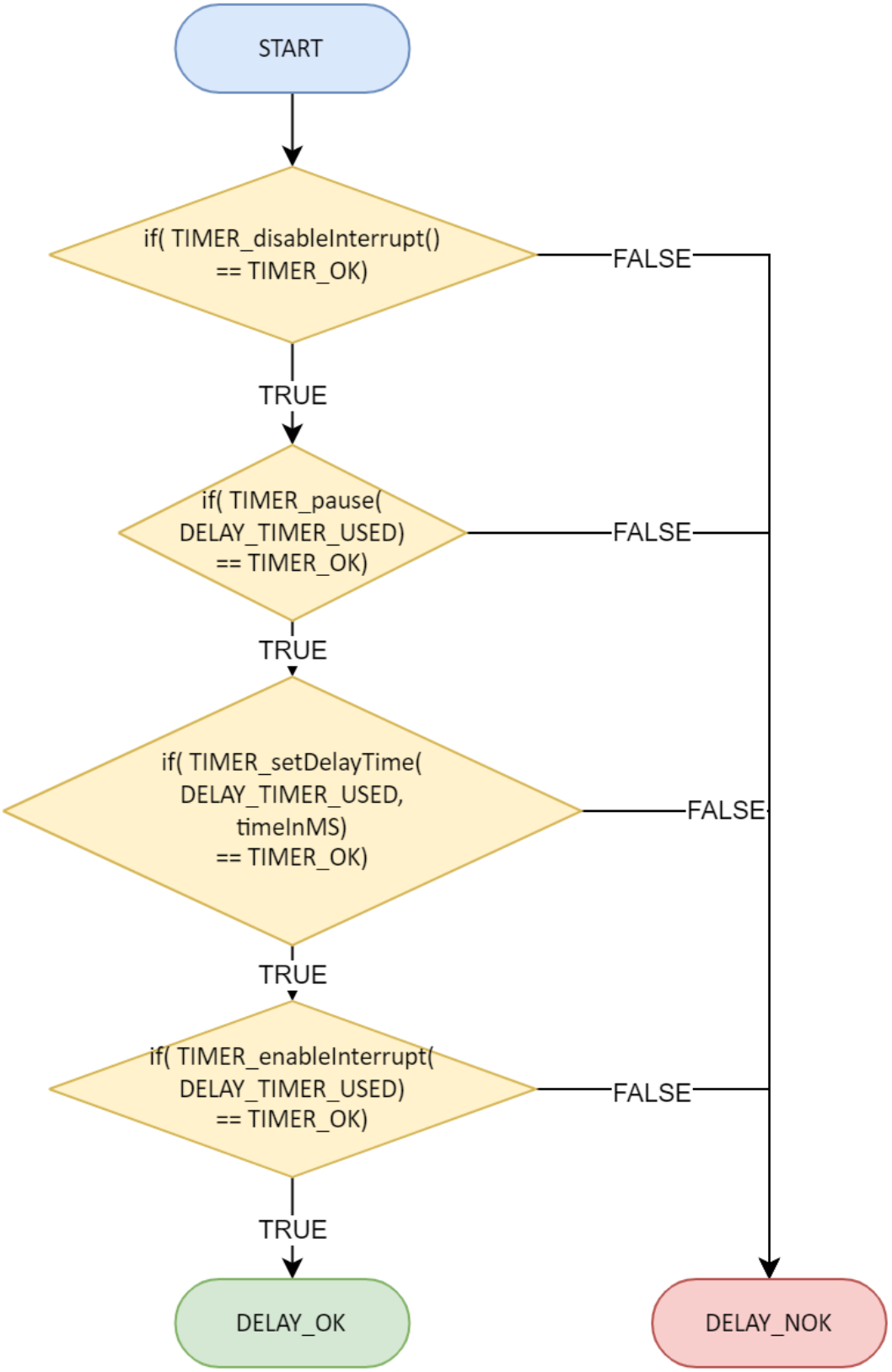


3.2.6. Delay Module

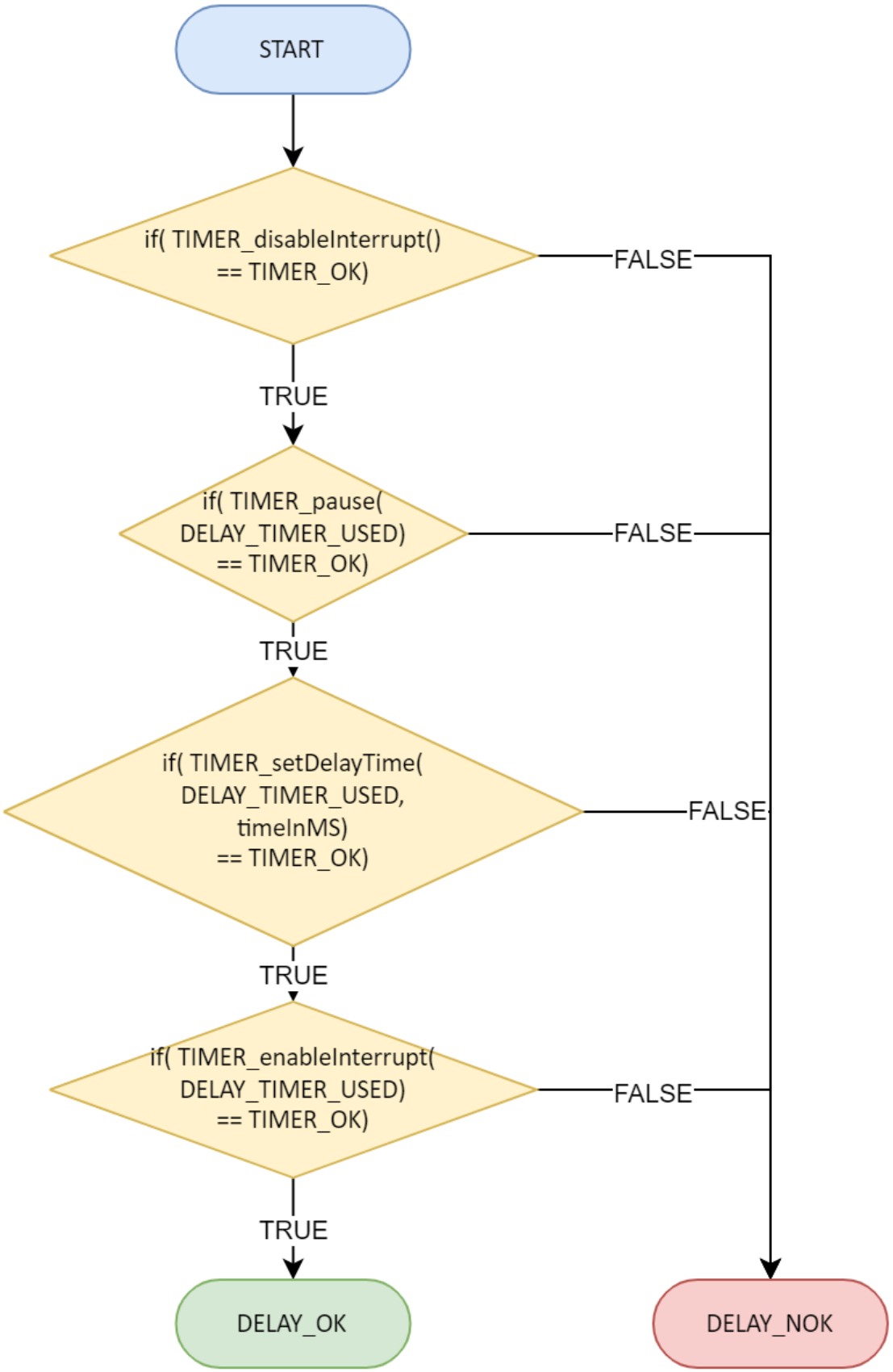
3.2.6.1 DELAY_init



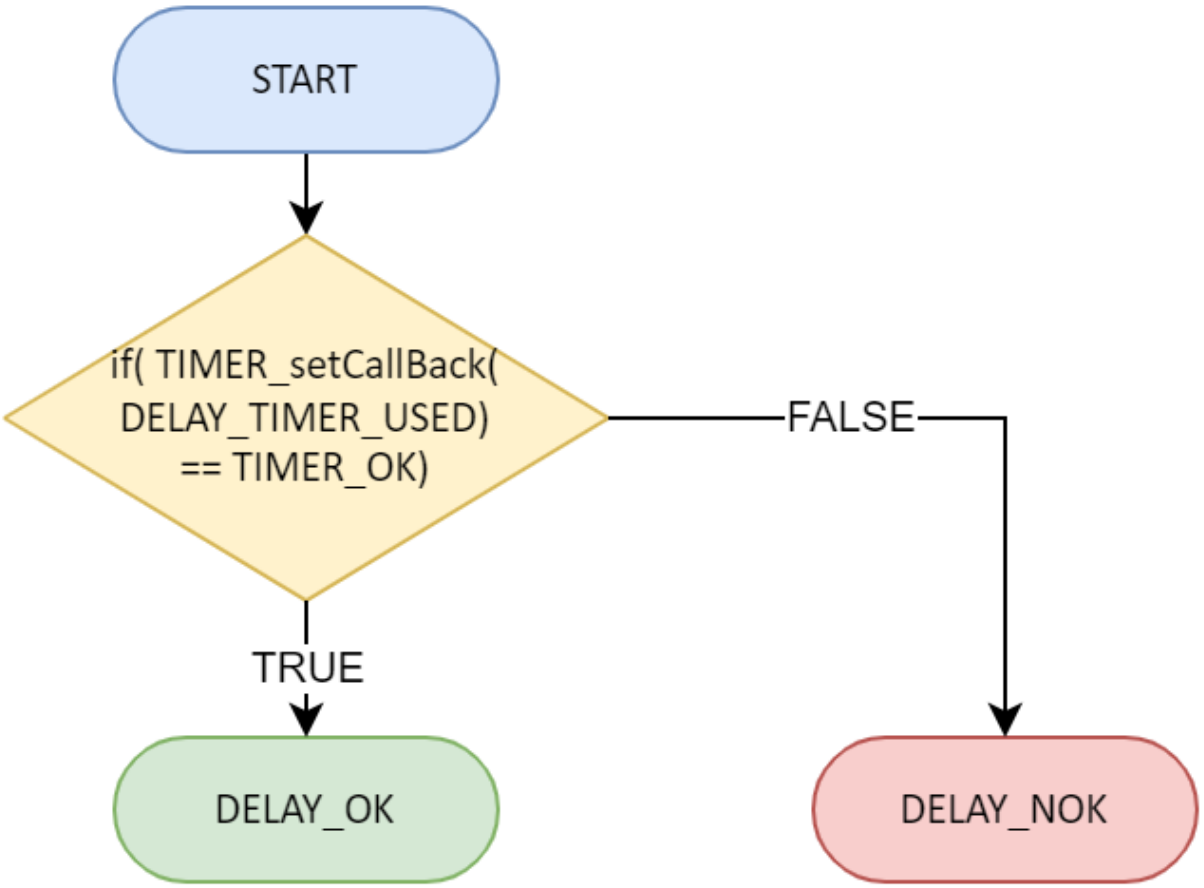
3.2.6.2 DELAY_setTime



3.2.6.3 DELAY_setTimeBlocking

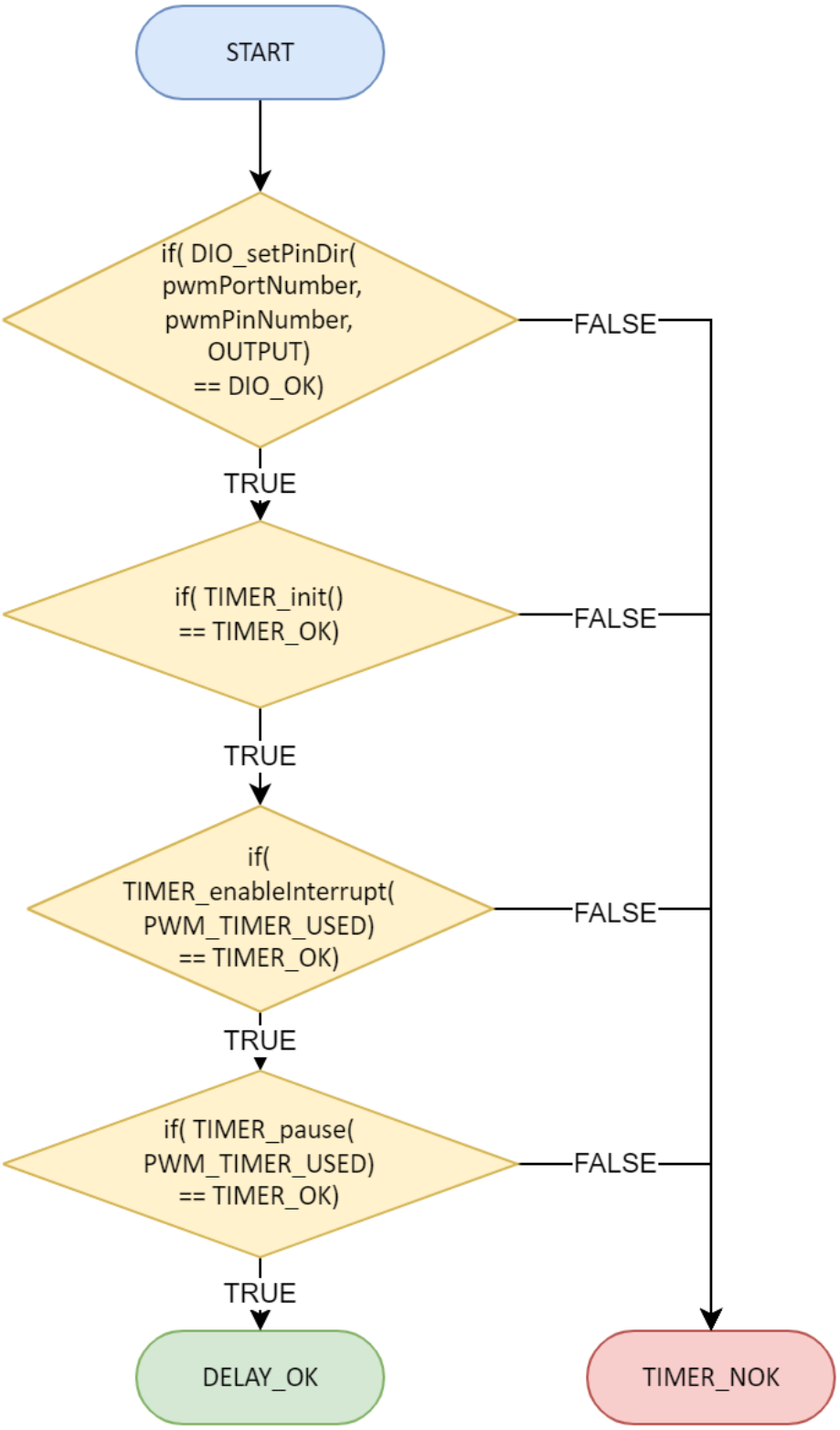


3.2.6.4 DELAY_setTimeBlocking

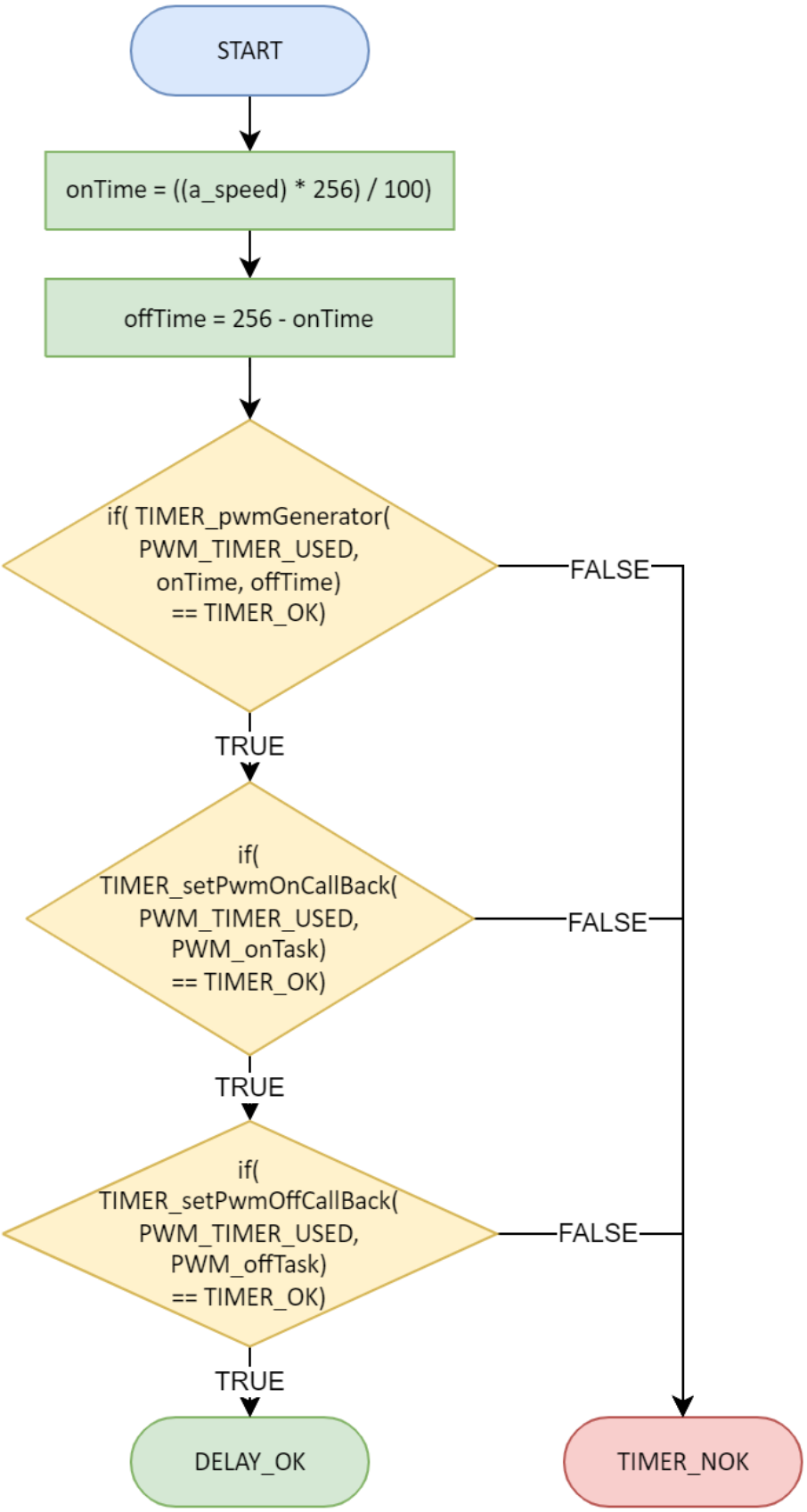


3.2.7 PWM Module

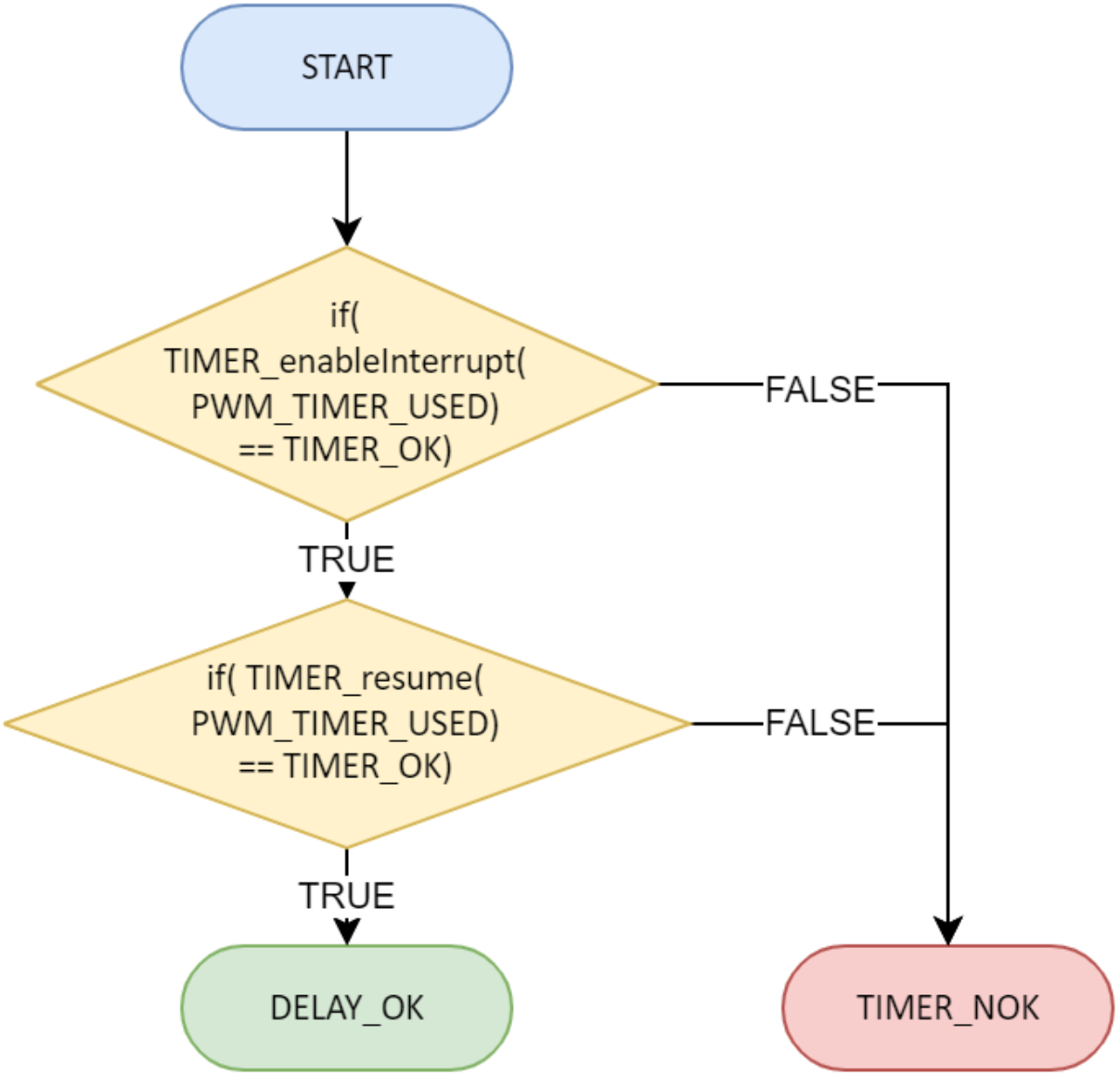
3.2.7.1. PWM_init



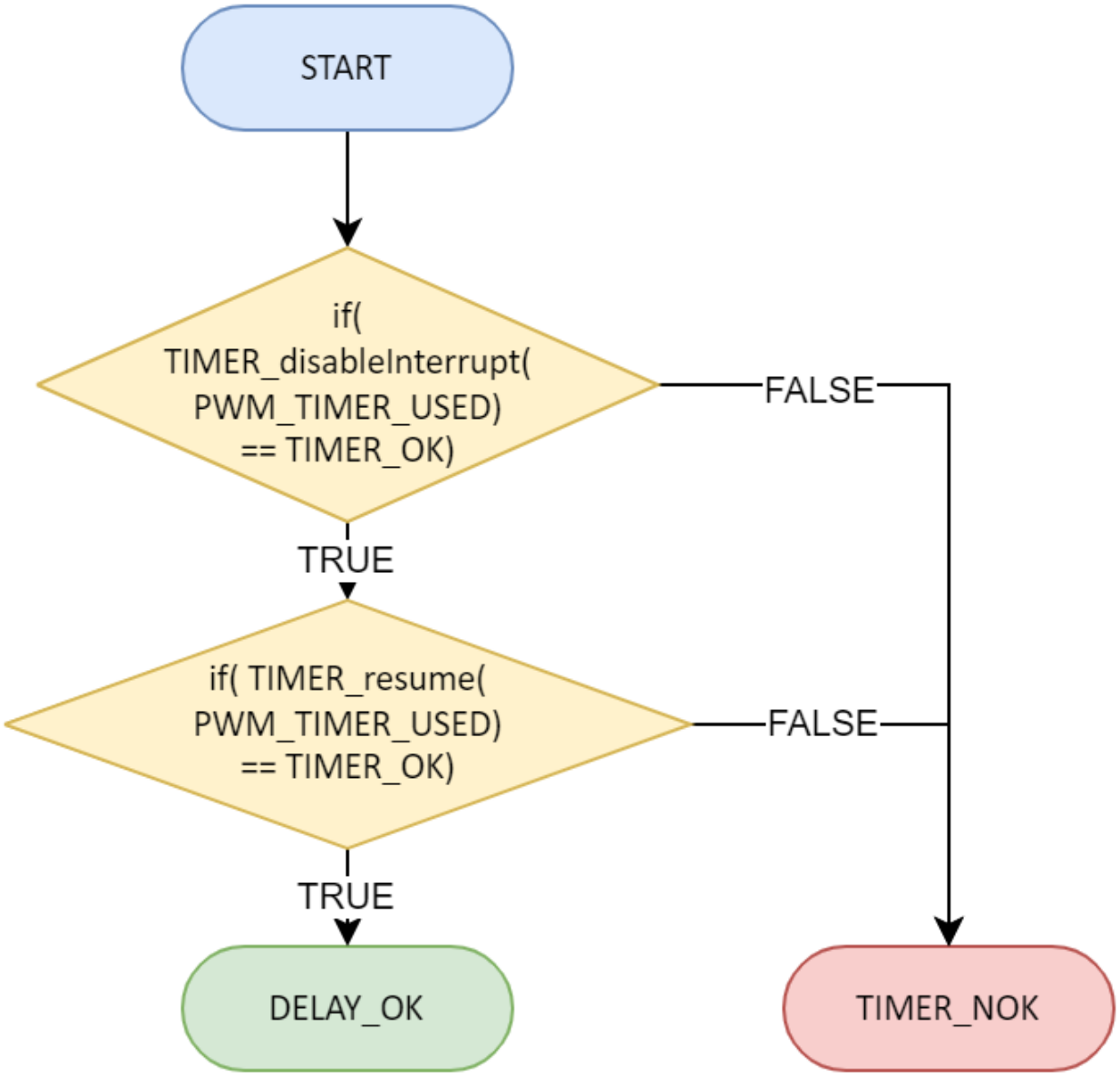
3.2.7.2. PWM_setDutyCycle



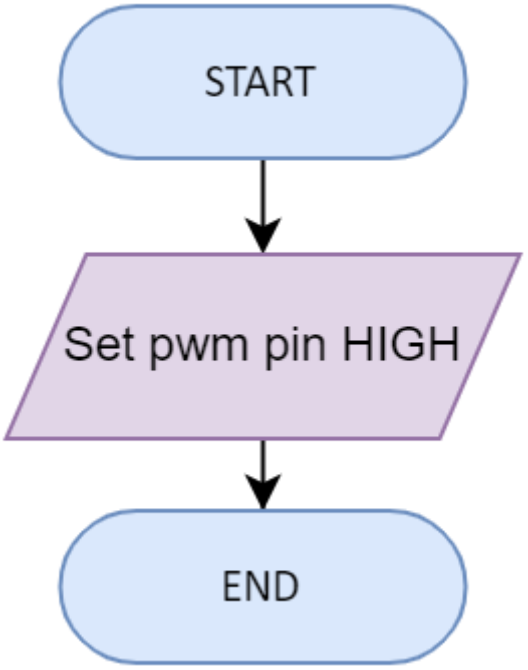
3.2.7.3. PWM_start



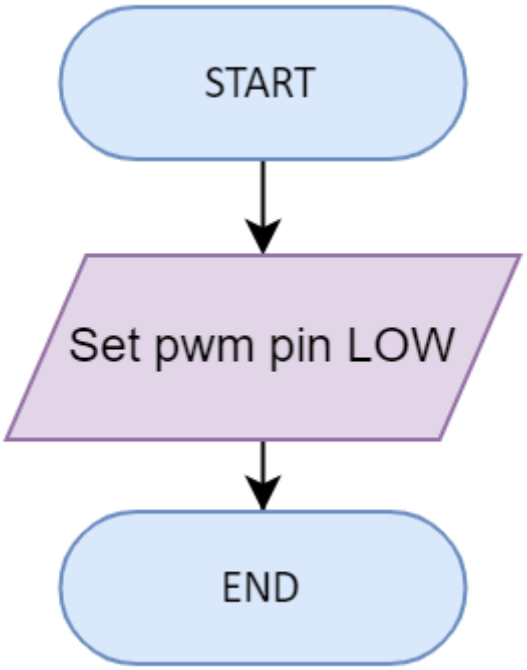
3.2.7.4. PWM_stop



3.2.7.5. PWM_onTask

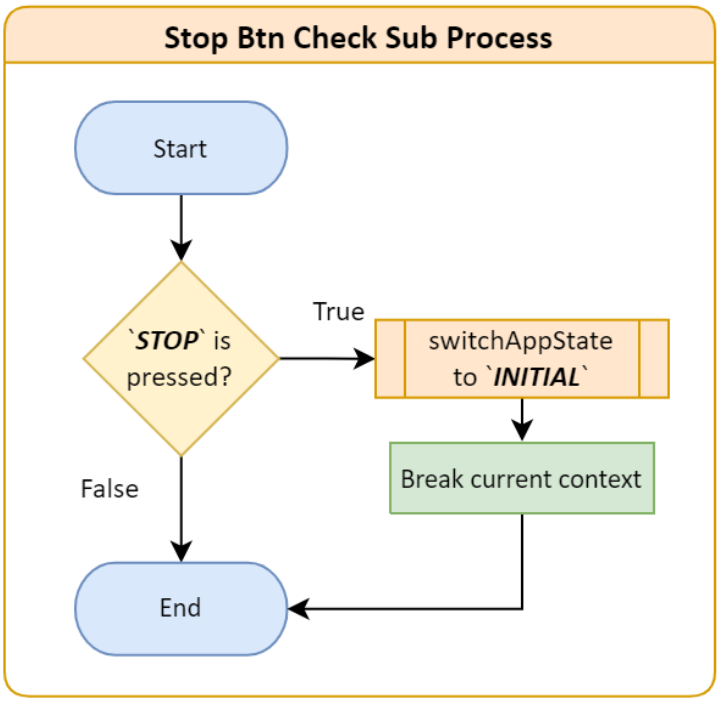


3.2.7.6. PWM_offTask

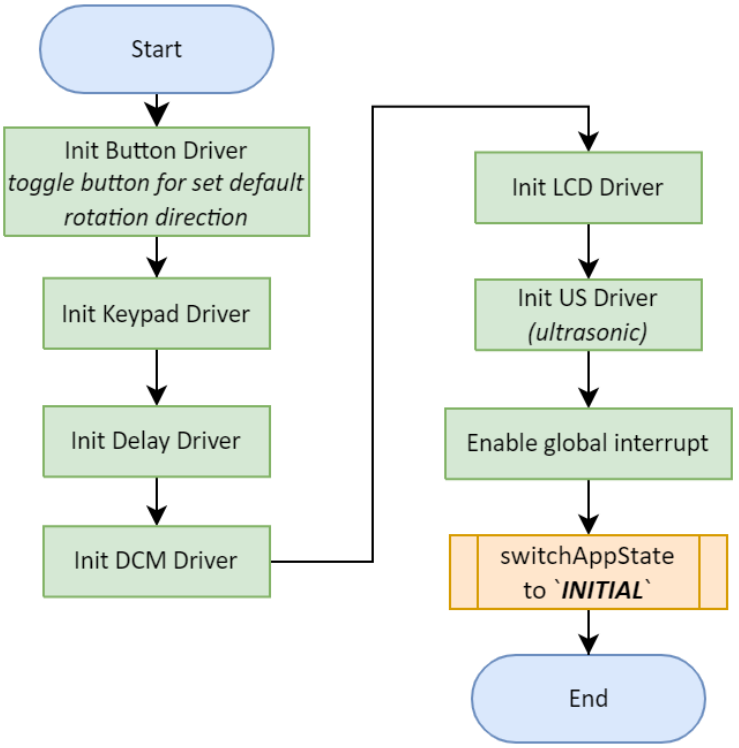


3.3. APP Layer

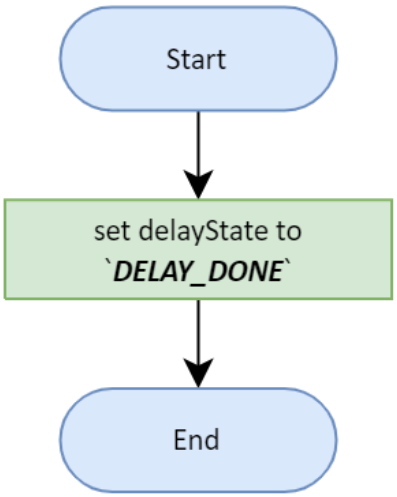
3.3.a. Stop Btn Check (sub-process)



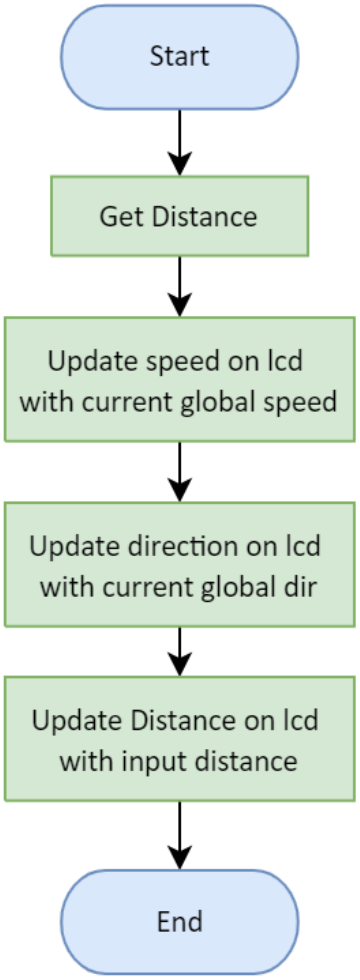
3.3.1. APP_initialization



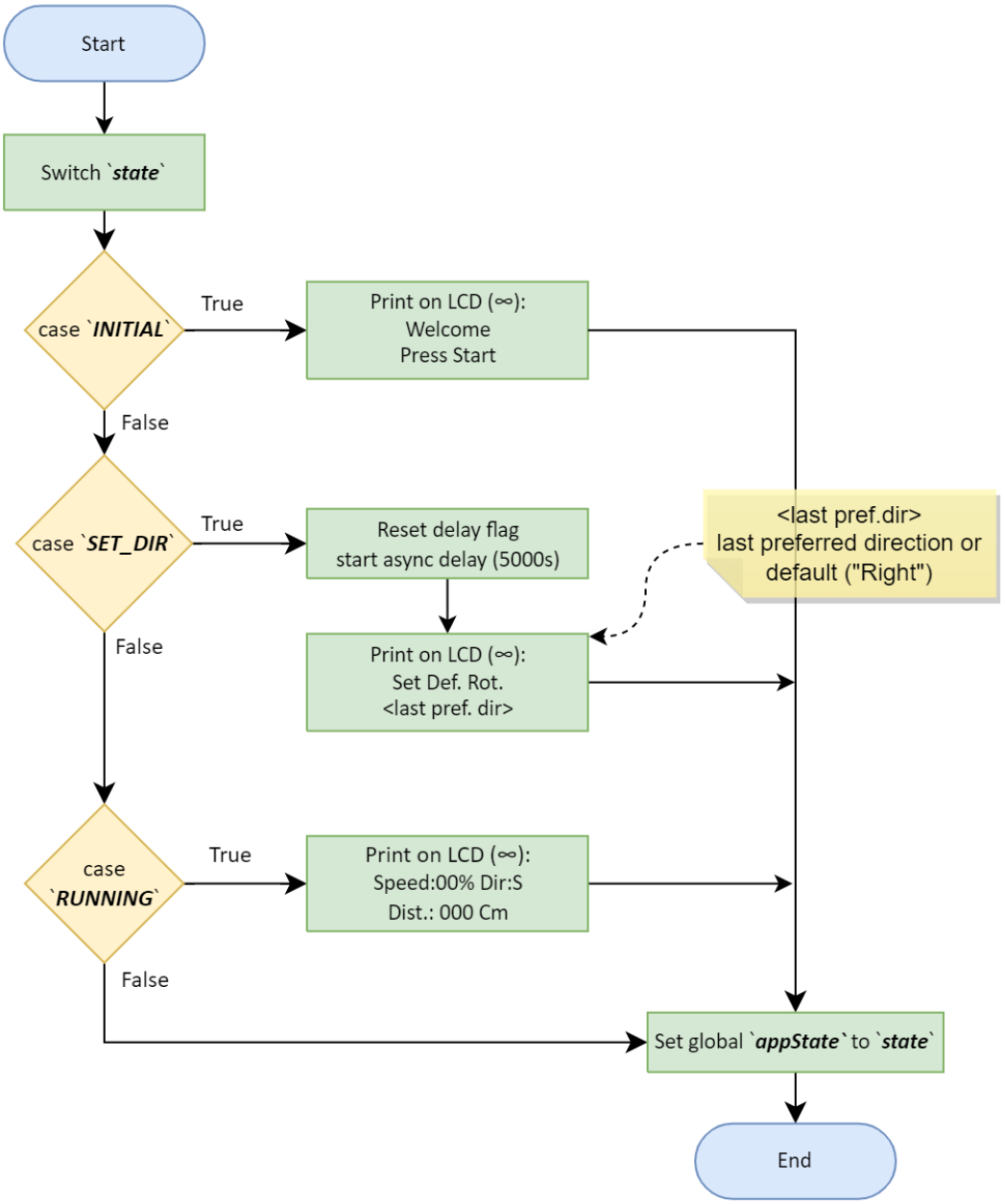
3.3.2. APP_delayNotification



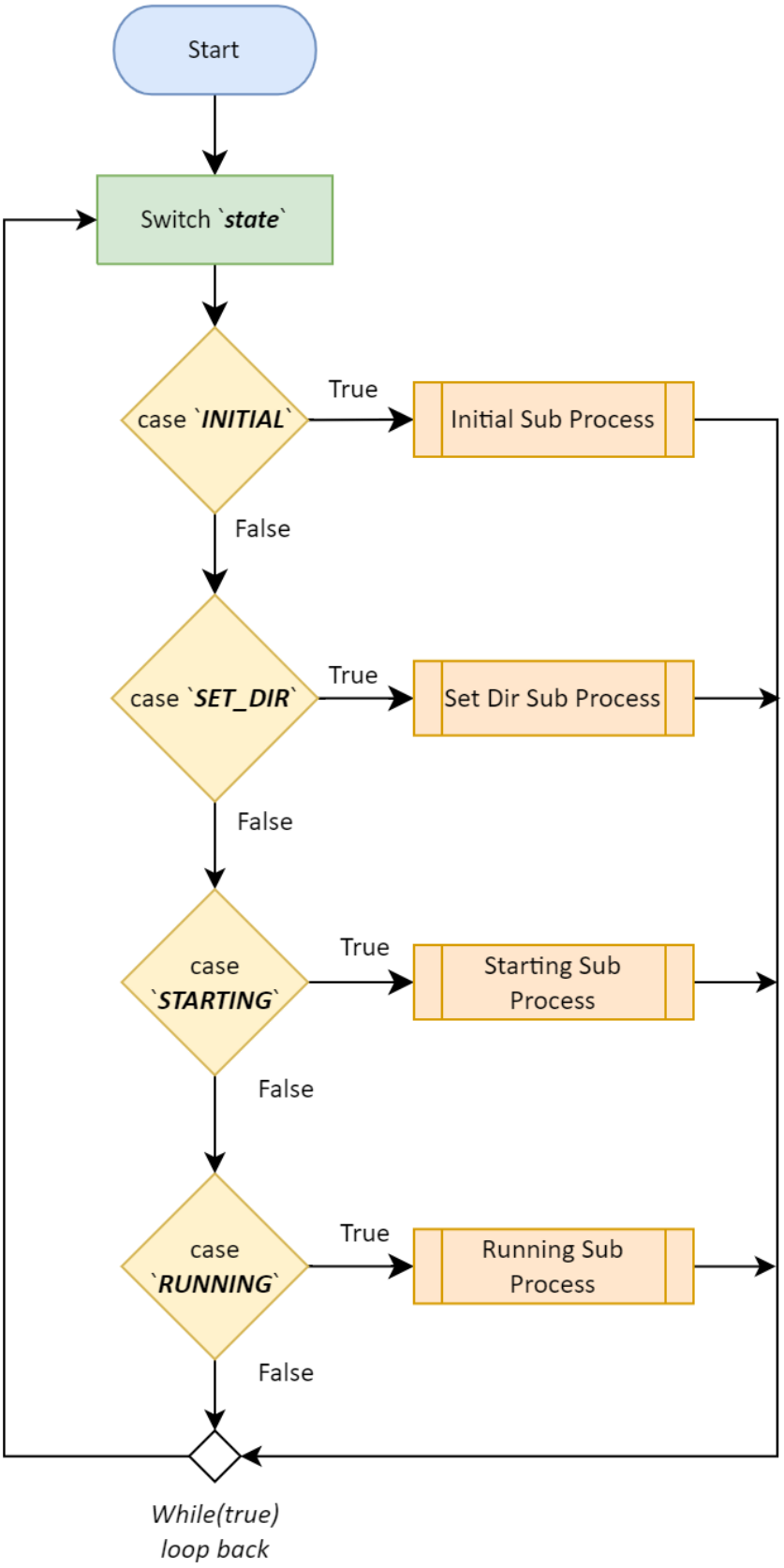
3.3.3. APP_updateUI



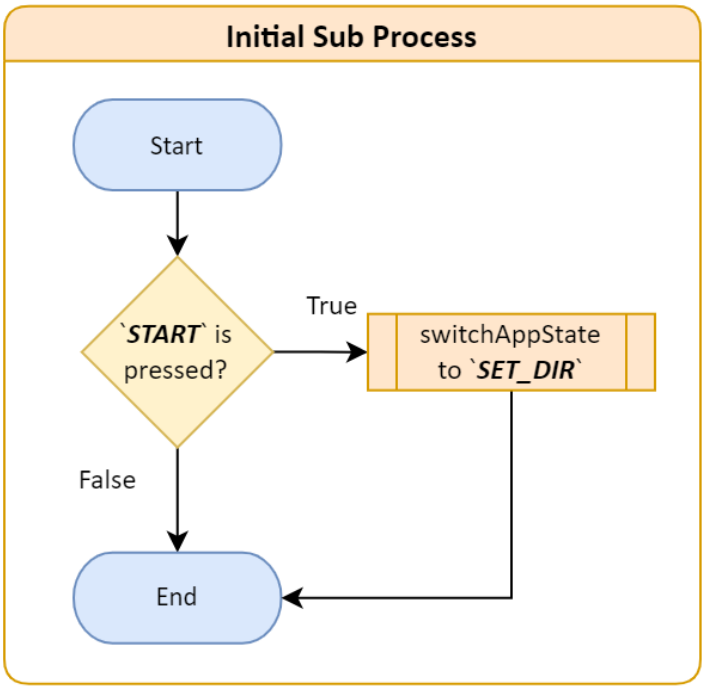
3.3.4. APP_switchState



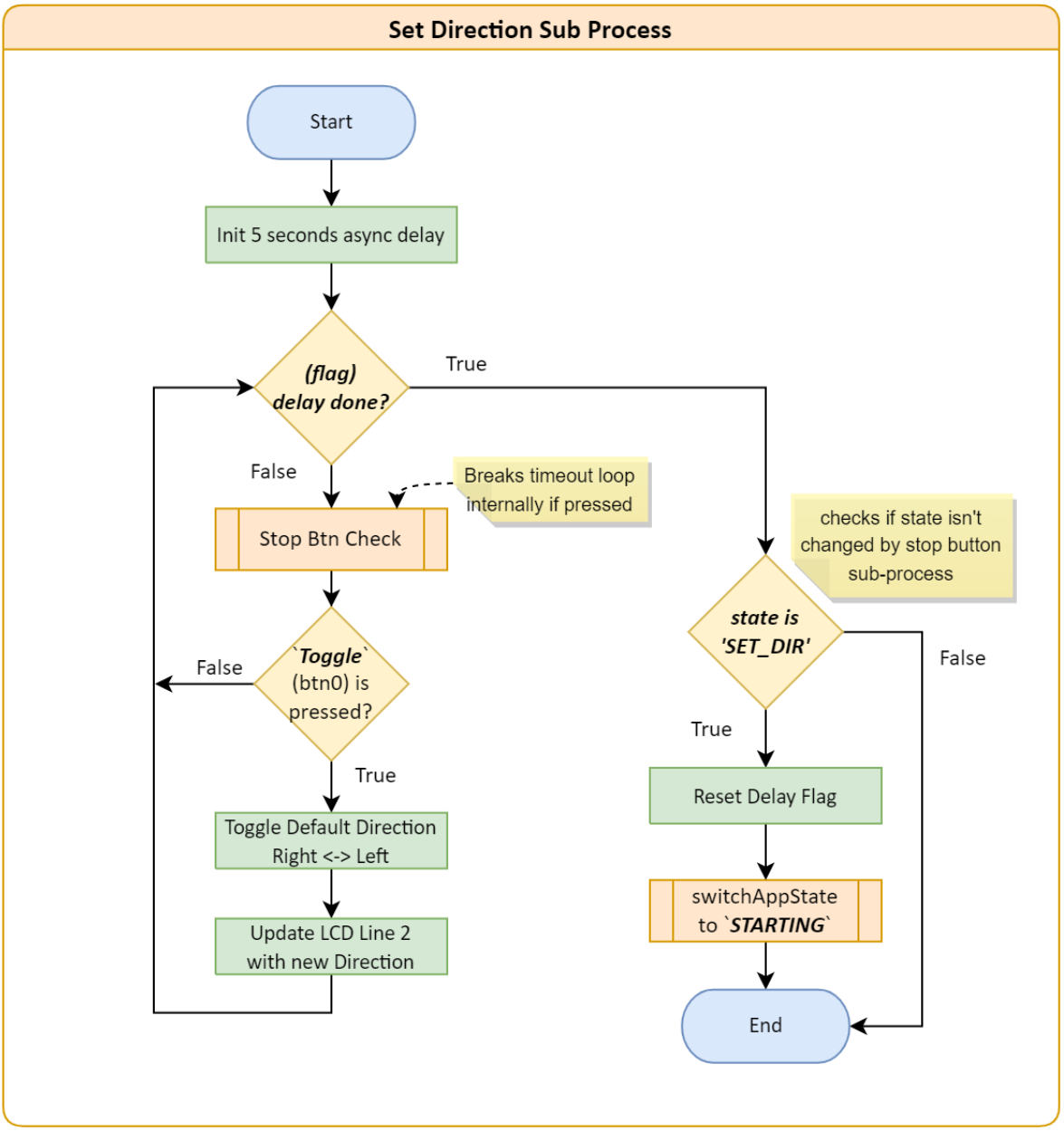
3.3.5. APP_startProgram



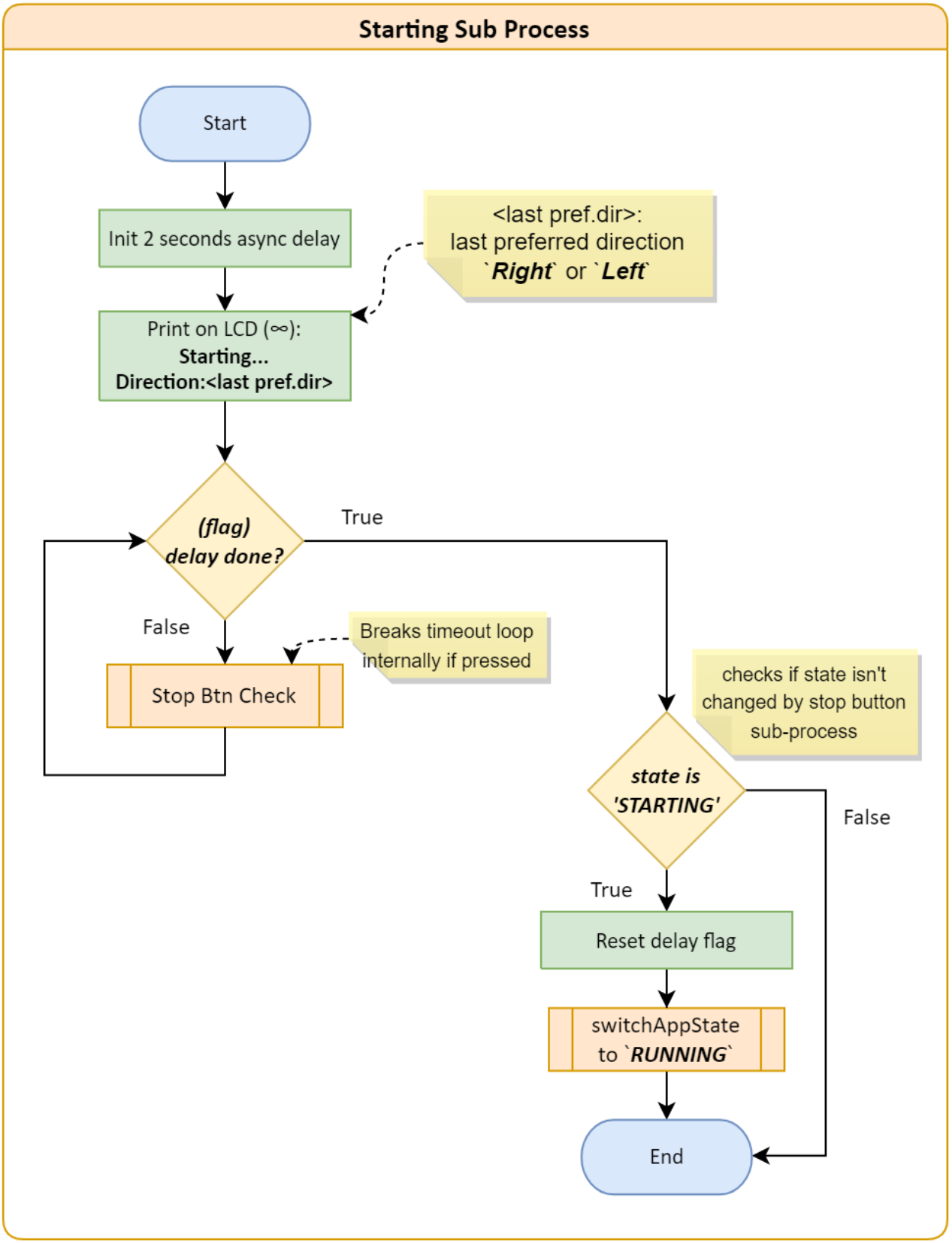
3.3.5.a. Initial (sub-process)



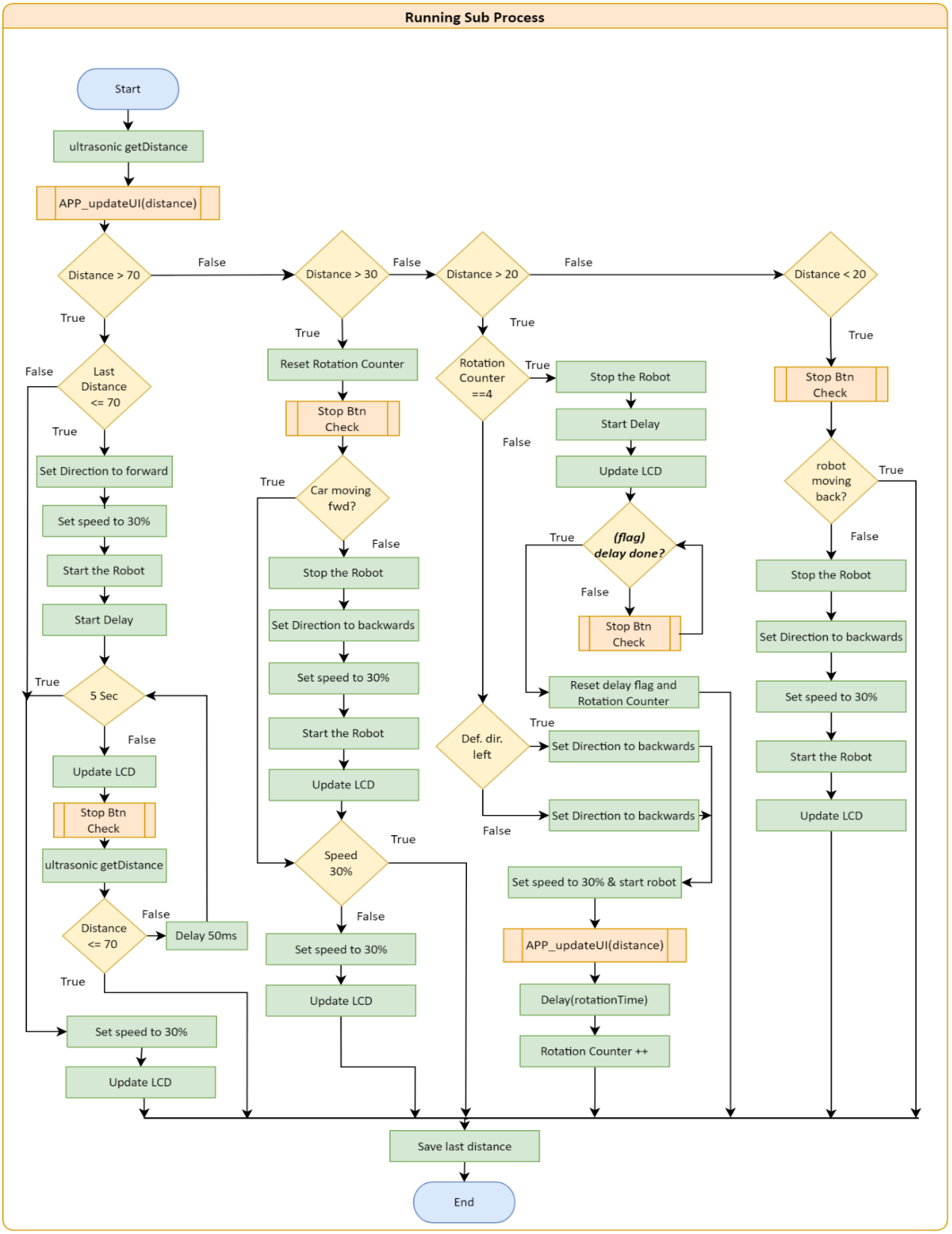
3.3.5.b. Set Direction (sub-process)



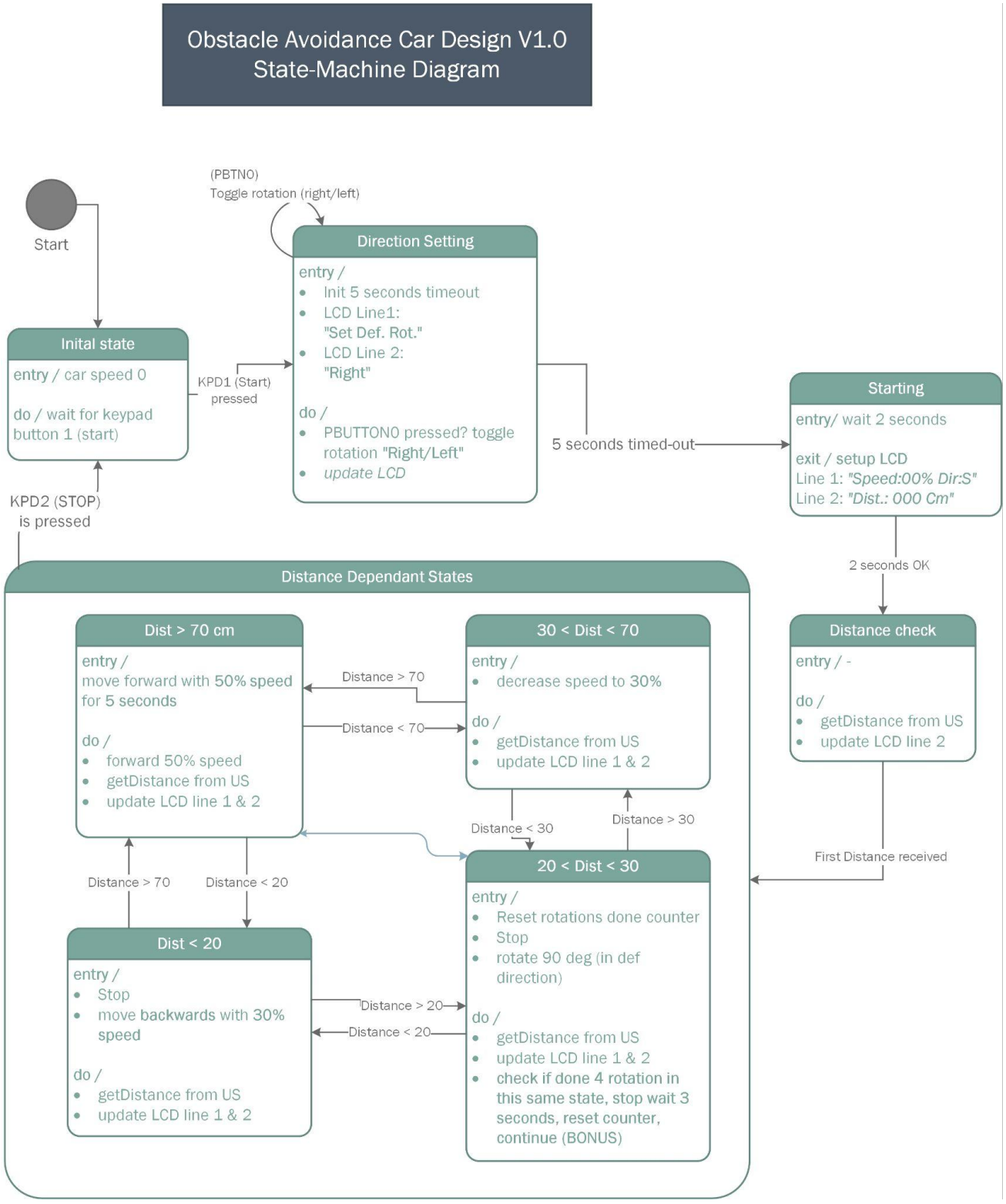
3.3.5.c. Starting (sub-process)



3.3.5.d. Running (sub-process)



3.3.b. App State Diagram *(for HQ click me)*



4. Pre-compiling and linking configurations

4.1. EXI Driver

4.1.1. Linking Configuration

```
typedef enum
{
	EXTI0,
	EXTI1,
	EXTI2
}en_EXI_num_t;

typedef enum
{
	EXI_DISABLE ,
	EXI_ENABLE
}en_EXI_state_t;

typedef enum
{
	LOW_LEVEL,
	ON_CHANGE,
	FALLING_EDGE,
	RISING_EDGE
}en_EXI_senseMode_t;

typedef enum
{
	EXI_OK,
	EXI_ERROR
}en_EXI_error_t;

typedef struct
{
	en_EXI_num_t  EXI_NUM;
	en_EXI_senseMode_t SENSE_MODE;
	en_EXI_state_t	EXI_EN;
}st_EXI_config_t;
```

```
|
| Options:
|                   .EXI_NUM =    EXTI0
|                                 EXTI1
|                                 EXTI2
|
|                   .SENSE_MODE = LOW_LEVEL          [for EXTI0 & EXTI1 only]
|                                 ON_CHANGE          [for EXTI0 & EXTI1 only]
|                                 FALLING_EDGE
|                                 RISING_EDGE
|
|                   .EXI_EN       = EXI_ENABLE
|                                         EXI_DISABLE
|
const st_EXI_config_t arr_g_exiConfigs[EXI_PINS_NUM] =
{
      {
            .EXI_NUM= EXTI0,
            .SENSE_MODE = RISING_EDGE,
            .EXI_EN = EXI_DISABLE
      },

      {
            .EXI_NUM= EXTI1,
            .SENSE_MODE = FALLING_EDGE,
            .EXI_EN = EXI_DISABLE
      },

      {
            .EXI_NUM= EXTI2,
            .SENSE_MODE = RISING_EDGE,
            .EXI_EN = EXI_DISABLE
      }
};
```

4.2. Timer Driver

4.2.1. Linking configurations

```
/**********************************************************************************
 *  GLOBAL DATA
 *********************************************************************************/
 const st_TIMER_config_t st_TIMER_config [NUMBER_OF_TIMERS_USED] =
{
/*    TIMER_number,     waveformUsed,     prescalerUsed     */
      {TIMER_0,         TIMER_OV,         TIMER_PRESCLNG_64},
      {TIMER_1,         TIMER_OV,         TIMER_PRESCLNG_64},
      {TIMER_2,         TIMER_OV,         TIMER_PRESCLNG_64}
};
```

4.2.2 Pre-compiled Configurations

```
/**********************************************************************************
 * GLOBAL CONSTANT MACROS
 *********************************************************************************/

/********************_SYSTEM_OSCILLARTOR_CLOCK_FREQUENCY_********************/
/*
 *    Enter microcontrollers frequency in Hz writing UL besides it
 */
#define F_CPU                                          8000000UL

/************************_NUMBER_OF_TIMERS_USED_************************/
/*
 *    number of timers used
 */
#define NUMBER_OF_TIMERS_USED                          3
```

4.3. ICU Driver

4.3.1. Linking Configuration

```
/* Predefined Pin options */
st_ICU_capturePins_t st_ICU_predefinedPins = {
        .PORT_D_PIN_2 = {
                .capturePin = PIN_2,
                .capturePort = PORT_D,
                .interruptNo = INT0
        },
        .PORT_D_PIN_3 = {
                .capturePin = PIN_3,
                .capturePort = PORT_D,
                .interruptNo = INT1
        },
        .PORT_B_PIN_2 = {
                .capturePin = PIN_2,
                .capturePort = PORT_B,
                .interruptNo = INT2
        }
};

/* Configuration */
static st_ICU_config_t st_gs_icuConfig = {
    // Capture Pin
       .icuCapturePin = PORT_B_PIN_2,

    // Callback Function on time received
       .timeReceivedCallbackFun = NULL // callback function
};

/* Functions */
st_ICU_config_t ICU_getConfig()
{
    switch (st_gs_icuConfig.icuCapturePin) {
        case PORT_D_PIN_2:
            st_gs_icuConfig.icuCapturePinData = st_ICU_predefinedPins.PORT_D_PIN_2;
            break;
        case PORT_D_PIN_3:
            st_gs_icuConfig.icuCapturePinData = st_ICU_predefinedPins.PORT_D_PIN_3;
            break;
        case PORT_B_PIN_2:
            st_gs_icuConfig.icuCapturePinData = st_ICU_predefinedPins.PORT_B_PIN_2;
            break;
    }
       return st_gs_icuConfig;
}
```

```
void ICU_setConfig(st_ICU_config_t st_a_icuConfig)
{
    st_gs_icuConfig = st_a_icuConfig;
}
```

4.4. PWM Driver

4.4.1. Linking Configurations

```
/*******************************************************************************
 *  GLOBAL DATA
 *******************************************************************************/
 const st_PWM_config_t st_PWM_config [NUMBER_OF_PWM_PINS] =
{
/*     PWM_#  PORT0_#,  PIN0_#*/
      {PWM_0, DIO_PORTD, DIO_PIN_4},
      {PWM_1, DIO_PORTD, DIO_PIN_5}
};
```

4.4.2. Pre-compiled Configurations

```
/*********************_SYSTEM_OSCILLARTOR_CLOCK_FREQUENCY_*******************/
/*
 *    Enter microcontrollers frequency in Hz writing UL besides it
 */
#define F_CPU                                          8000000UL

/*****************************_TIMER_USED_*********************************/
/*
 *    Timer used to generate delay
 */
#define PWM_TIMER_USED                                 TIMER_0

/************************_NUMBER_OF_PWM_PINS_*************************/
/*
 *    Number of pwm pins used to generate PWM signal on them
 */
#define NUMBER_OF_PWM_PINS                             2
#define PWM_0                                          0
#define PWM_1                                          1
```

4.5. Delay Driver

4.5.1. Pre-compiled Configurations

```
/*******************************************************************************
 *  GLOBAL CONSTANT MACROS
 *******************************************************************************/
/************************_SYSTEM_OSCILLARTOR_CLOCK_FREQUENCY_***************/
/*
 *    Enter microcontrollers frequency in Hz writing UL besides it
 */
#define F_CPU                                           8000000UL

/************************_TIMER_USED_*************************/
/*
 *    Timer used to generate delay
 */
#define DELAY_TIMER_USED                                TIMER_2
```

4.6. DCM Driver

4.6.1. Linking Configurations

```
/*******************************************************************************
*  GLOBAL DATA
*******************************************************************************/
const st_DCM_config_t st_DCM_config [NUMBER_OF_DCMS_USED] =
{
/*     DCM_#, PORT0_#,    PIN0_#*, PORT1_#,    PIN1_# */
      {DCM_0, DIO_PORTD, DIO_PIN_2, DIO_PORTD, DIO_PIN_3},
      {DCM_1, DIO_PORTD, DIO_PIN_6, DIO_PORTD, DIO_PIN_7}
};
```

4.6.2. Pre-compiled Configurations

```
/*******************************************************************************
 *  GLOBAL CONSTANT MACROS
 *******************************************************************************/
/************************_NUMBER_OF_DCMS_USED_************************/
/*
 *    number of dcms used
 */

#define NUMBER_OF_DCMS_USED              2
```

4.8. US (ultrasonic) Driver

4.8.1. Linking Configuration

```
static st_US_config_t st_gs_usConfig = {
      US_PORT_B,
    US_PIN_3,
    US_PIN_2,
};

st_US_config_t US_getConfig()
{
      return st_gs_usConfig;
}

void US_setConfig(st_US_config_t st_a_usConfig)
{
    st_gs_usConfig = st_a_usConfig;
}
```

4.9. LCD Driver

4.9.1. pre-compiling configuration

```
#define LCD_8_BIT_MODE                  0
#define LCD_4_BIT_MODE                  1

#define LCD_POS_0 0
#define LCD_POS_1 1
#define LCD_POS_2 2
#define LCD_POS_3 3
#define LCD_POS_4 4
#define LCD_POS_5 5
#define LCD_POS_6 6
#define LCD_POS_7 7
#define LCD_POS_8 8
#define LCD_POS_9 9
#define LCD_POS_10 10
#define LCD_POS_11 11
#define LCD_POS_12 12
#define LCD_POS_13 13
#define LCD_POS_14 14
#define LCD_POS_15 15

#define LCD_ROW_0 0
#define LCD_ROW_1 1

#define LCD_MAX_U32_DIGITS 10
```

4.10. KPD Driver

4.10.1. pre-compiling configuration

```
/** Macros **/

#define LOW 0
#define HIGH 1
#define OUTPUT 0
#define INPUT 1

#define KPD_DEBOUNCE_DELAY 20

/** User-defined data types **/
#define KEYPAD_PORT DIO_PORTC

#define KEYPAD_COLUMN_0 DIO_PIN_5
#define KEYPAD_COLUMN_1 DIO_PIN_6


#define KEYPAD_ROW_0 DIO_PIN_2
#define COL_INIT 5
#define COL_FINAL 6

#define ROW_INIT   2
#define ROW_FINAL  2

#define KPD_KEY_NOT_PRESSED 0
#define KPD_KEY_START 1
#define KPD_KEY_STOP 2
```

4.11. BTN Driver

4.11.1. pre-compiling configuration

```
/* Macros */
#define MAX_PIN_NUMBER  7
#define MAX_PORT_NUMBER 3

#define BTN_DELAY_BTN_DEBOUNCE      50
```

5. References

1. Draw IO
2. Layered Architecture | Baeldung on Computer Science
3. Microcontroller Abstraction Layer (MCAL) | Renesas
4. Hardware Abstraction Layer - an overview | ScienceDirect Topics
5. What is a module in software, hardware and programming?
6. Embedded Basics – API's vs HAL's
7. 4WD Complete Mini Plastic Robot Chassis Kit - With 4x 90 Degree Motors - RAM
8. Moving-Car-Project: Embedded C application controlling a four-driving wheel robot, and moving it in a rectangular shape. (github.com)
9. ultrasonic sensor library in proteus - projectiot123 Technology Information Website worldwide